Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Andrejc

Uporaba JavaEE mikrostoritev pri razvoju celovite rešitve za upravljanje s spletnimi oglasi

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Obstaja veliko spletnih mest, kjer lahko potencialni kupec išče želeni predmet. Obilica spletnih mest omogoča izbiro med raznolikimi izdelki, je pa časnovno potratno večkratno pregledovanje teh spletnih strani in vsakokratno določanje kriterijev za prikaz izdelkov. V okviru diplomskega dela je vaša naloga naloga razviti rešitev, ki poenostavi in predvsem pohitriti celoten proces ponavljajočega iskanja izbranega izdelka na različnih spletnih portalih. Pri implementaciji rešitve se osredotočite na funkcijo sprotnega periodičnega preverjanja spletnih portalov in obveščanja uporabnikov o novih oglasih oziroma o spremenjenih oglasih, glede na uporabnikove kriterije. Pri tem realizirajte tako odjemalski del kot tudi zaledni del rešitve. V okviru odjemalskega dela implementirajte spletno aplikacijo poleg tega pa za zagotavljanje čim boljše uporabniške izkušnje razivjte tudi razširitev za spletni brskalnik ter mobilno aplikacijo. Pri zalednem delu pa se osredotočite na pregledovanje spletnih portalov. Pregledovanje spletnih portalov realizirajte z uporabo mikrostoritev. V okviru naloge pa celovito rešitev tudi testirajte in predstavite odzive uporabnikov na razvito rešitev.

Mentorju, doc. dr. Alešu Smrdelu, se zahvaljujem za pomoč pri pisanju in zaključevanju diplomske naloge. Zahvalo bi namenil tudi vsem bližnjim, ki so mi stali ob strani pri tem zaključku dodiplomskega študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
REST	Representational State Transfer	predstavitveni prenos stanja
JavaEE	Java Enterprise Edition	poslovna izdaja Jave
JavaSE	Java Standard Edition	standardna izdaja Jave
EJB	Enterprise Java Beans	javanska strežniška zrna
JDBC	Java Database Connectivity	javanska povezljivost z zbirkami podatkov
WORA	Write Once, Run Everywhere	Piši enkrat, poganjaj vsepovsod
JVM	Java Virtual Machine	javanski navidezni stroj
API	Application Programming Interface	aplikacijski programski vmesnik
ORDBMS	Object-Relational Database Management System	sistem za upravljanje z relacijskimi podatkovnimi bazami
JSON	JavaScript Object Notation	JavaScript zapis objektov
SQL	Structured Query Language	strukturiran povpraševalni jezik za delo s podatkovnimi bazami
JPA	Java Persistence API	Java API za manipuliranje s podatkovnimi bazami
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
HMAC	Hash Message Authentication Code	zgoščena avtentikacijska koda sporočila
SHA	Secure Hash Algorithm	algoritem za varno zgoščevanje
JAR	Java Archive	Java arhiv

Povzetek

Naslov: Uporaba JavaEE mikrostoritev pri razvoju celovite rešitve za upravljanje s spletnimi oglasi

Avtor: Gašper Andrejc

Diplomska naloga govori o procesu razvijanja celovite rešitve za sledenje in upravljanje s spletnimi oglasi, kot pomoč uporabniku pri nakupu želenega izdelka. Končni izdelek diplomske naloge je rešitev, ki je sestavljena iz spletne aplikacije, mobilne aplikacije, razširitve v spletnem brskalniku ter zalednega sistema. Diploma poleg idejne zasnove predstavlja tudi uporabo JavaEE mikrostoritev in ostalih tehnologij ter argumentira, zakaj so izbrani okvirji najprimernejši za takšno rešitev. Cilj diplomske naloge je bil izdelati celovito rešitev, s katero bo nakupovanje rabljenih predmetov na spletu enostavnejše in učinkovitejše. Za enostavno povezovanje omenjenih odjemalskih delov rešitve ter dopuščanja odprtih možnosti pri nadgrajevanju, je osrednji del aplikacije zgrajen iz mikrostoritev, kar poleg omenjenih prednosti omogoča tudi skaliranje različno obremenjenih delov infrastrukture. Težnja je predvsem dodobra razviti zaledni del aplikacije, ne pa tudi vseh odjemalskih sistemov. Pri teh se osredotočamo predvsem na funkcionalnost in ne na estetiko, saj želimo bolj kot uporabniški vmesnik predstaviti sobivanje vseh teh sistemov celovite rešitve. Na koncu diplomske naloge predstavimo in opišemo napake, ki so bile storjene ob razvoju, ter težave, s katerimi smo se spopadali, ko smo aplikacijo prvič postavili pred uporabnike.

Ključne besede: spletni oglasi, mikrostoritve, JavaEE, spletni portal, apli-

kacija, razširitev.

Abstract

Title: Use of JavaEE microservices in implementation of online advertisements manipulation

Author: Gašper Andrejc

The following thesis talks about developing a complete solution for following and manipulation of online advertisements. The end product of the thesis is comprised of web application, mobile application, browser extension and a backend system. Besides the conceptual design the thesis also presents the use of JavaEE microservices and other technologies, and also argues why the selected frame is the best for such a solution. Main goal of the thesis is to provide the end user with a solution, which allows for easier and more efficient buying of an used item. For easier development and potential upgrading of the solution in the future, the whole backend system is built using microservices, which besides mentioned advantages also allows for scaling of unequally loaded parts of the infrastructure. The main effort is to thoroughly develop the backend of the system and not all of the client parts. For the client parts of the system we focus mainly on functionality and not on the aesthetics, since we wish to describe the cohabitation of all parts of the system more thorougly than the user interface. At the end of the thesis we also introduce and describe all of the mistakes, which were made during the development, and also the problems, which we dealt with, when we introduced the application to end users for the first time.

Keywords: online advertisements, microservices, JavaEE, web portal, ap-

plication, online, extension.

Poglavje 1

Uvod

V uvodnem poglavju diplomske naloge se sprehodimo skozi miselni proces, na katerem smo zasnovali idejo celovite rešitve. Poleg motivacijskega dela ter predstavitve, kako je diplomska naloga strukturirana, predstavimo še cilje, ki smo si jih zastavili pred pisanjem diplomske naloge in implementiranjem rešitve.

1.1 Motivacija za izdelavo diplomske naloge

Pri procesu nakupovanja rabljenega vozila, ko si želimo čim prej najti ugoden in hkrati dovolj dobro ohranjen jekleni konjiček, večkrat dnevno obiščemo različne spletne portale, kjer so objavljeni oglasi.

Sedanja najpopularnejša spletna mesta za prodajo in nakup rabljenih vozil imajo enormne baze uporabnikov, ki morajo za vsak pregled strani za želeno vozilo obiskati spletno stran ter vsakič znova vnašati nabor želenih filtrov za posamezno iskanje. To je lahko časovno zelo zamudno, še posebej, ko iščemo po več različnih sklopih filtrov. Z aplikacijo, ki je tema te diplomske naloge, bi za vsako želeno spletno stran z oglasi določili nabor filtrov po katerem bi aplikacija iskala in nas nato obveščala o morebitnih novih oglasih, ki bi se v tem času pojavili. Danes je hiter dostop do informacij ključnega pomena, kar še posebno velja za trg rabljenih vozil ali pa nakupovanje red-

kih zbirateljskih predmetov, kjer oglasi hitro najdejo kupca za oglaševani predmet.

1.2 Struktura diplomske naloge

V uvodnem delu diplomske naloge je predstavljena motivacija za izdelavo aplikacije oziroma naloge ter razlog zakaj in komu bi takšna aplikacija koristila. Poleg strukture je predstavljen še končni cilj, do katerega želimo v sklopu te diplomske naloge priti.

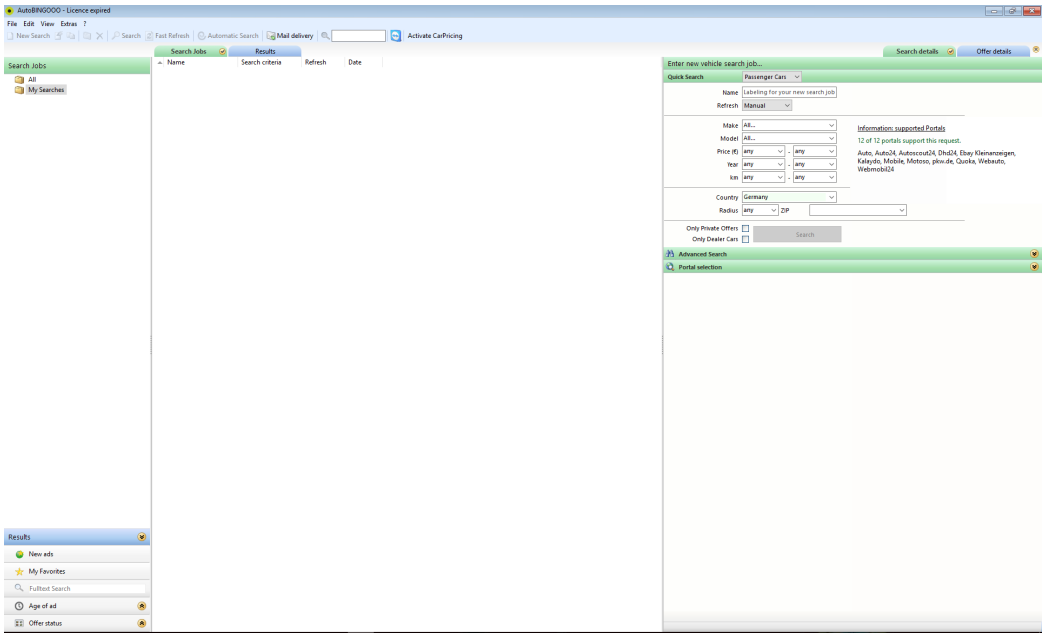
V drugem delu je predstavljen pregled in opis tehnologij, ki smo jih uporabili za razvoj vseh delov celovite rešitve. Sam razvoj je opisan v tretjem poglavju, ki se začne z opisom arhitekturnega dela infrastrukture, nato pa se razdeli na razvoj osrednjega dela aplikacije (strežniškega dela) in na razvoj vsakega izmed odjemalskih delov - spletne aplikacije, mobilne aplikacije ter razširitve v spletnem brskalniku Google Chrome. Na koncu tega poglavja so predstavljeni tudi problemi (in rešitve), s katerimi smo se srečali, ko smo aplikacijo postavili pred prve uporabnike.

Diplomsko delo zaključimo s predstavitvijo rezultatov, predstavljene so tudi ideje in cilji za prihodnost, ovrednotena pa je tudi uspešnost vzpostavitve celovite rešitve.

1.3 Pregled področja

Pred začetkom izdelave diplomske naloge smo pregledali tudi področje, na katero posegamo z našo celovito rešitvijo. Ugotovili smo, da aplikacije oziroma rešitve, ki bi omogočala, kar omogočamo naša rešitev, še ne obstaja ne na slovenskem ne na evropskem trgu. Potencialno primerljiva aplikacija bi bila aplikacija Auto bingooo [4], računalniški program, ki zbira podatke z večih spletnih portalov in jih statistično analizira, rezultati te analize pa so namenjeni predvsem zahtevnejšim uporabnikom. Slabost je nepreglednost, zahtevnejša in neintuitivna uporaba ter nasploh profesionalna orientiranost

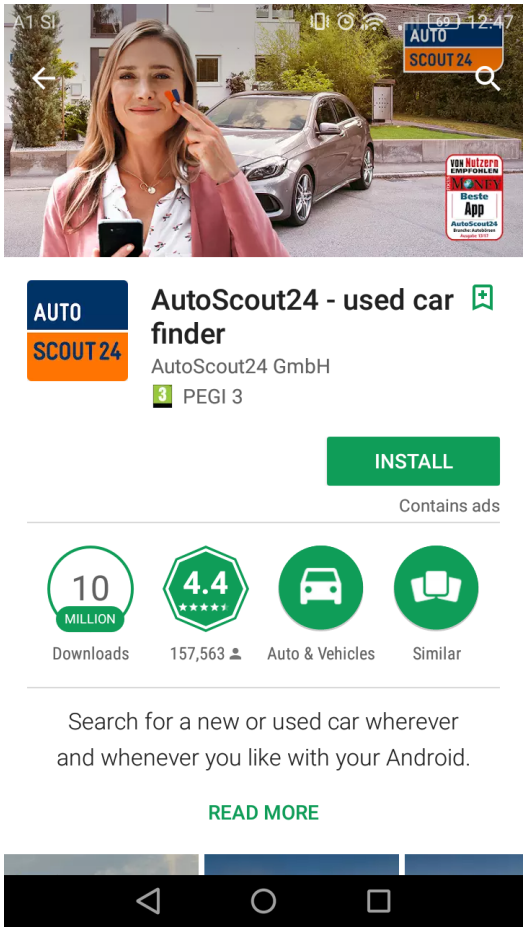
programa - torej cilja na povsem drugačne uporabnike (Slika 1.1).



Slika 1.1: Auto bingooo uporabniški vmesnik

Delno konkurenco predstavlja tudi spletni portal AutoScout24 [5] s svojimi lokalnimi izpeljankami, saj že ponujajo mobilno aplikacijo (Slika 1.2), ki ima med drugim tudi možnost sledenja kriterijem oziroma obveščanje, ko se pojavi novo vozilo. Slabost te rešitve je, da jo omogoča le mobilna aplikacija poleg tega pa tudi ni razširjena na več spletnih mest (le na AutoScout24).

Pomembne prednosti naše celovite rešitve so, da združujemo več portalov, uporabniški vmesnik pa je dovolj preprost tudi za laičnega uporabnika. Pri Auto bingooo aplikaciji vidimo problem predvsem pri nepreglednosti in odvečnosti podatkov. Uporabnik mora biti vešč tako v računalniškem smislu kot tudi v tem, kar kupuje (npr. poznati mora avtomobile). V našem primeru nudimo spletno aplikacijo (Auto bingooo je aplikacija, ki si jo je potrebno namestit na računalnik), katere uporabniški vmesnik je enostavnejši in bolj preprost. Od aplikacije AutoScout24 se razlikujemo tako, da združujemo več portalov in zato uporabnikom omogočimo preglednejšo sliko. Glavna prednost pa je ta, da poleg prevoznih sredstev omogočamo sledenje več kate-



Slika 1.2: Mobilna aplikacija AutoScout24, ki omogoča nekatere funkcionalnosti, ki jih ponujamo tudi sami

gorijam predmetov, in sicer omogočamo obvestila na več nivojih (SMS, prek elektronske pošte, prek mobilne aplikacije) in sodelovanje uporabnikov prek komentatorskih modulov.

Pred izdelavo smo izvedli tudi anketo (celotni rezultati ankete so predstavljeni v dodatku A) z nekaj potencialnimi uporabniki (pridobili smo 351 odgovorov). Ocenili smo, da lahko povprečnemu uporabniku prihranimo do 80 % časa, ki ga nameni iskanju oglasov, pri tem pa ohranimo ali še povečamo učinkovitost pridobivanja želenega predmeta. 79 % vseh anketirancev je odgovorilo, da pri iskanju rabljenega vozila uporablja spletne portale, od tega

jih je 22 % za posamični pregled vseh oglasov porabi več kot 15 minut. 19 % anketirancev je odgovorilo tudi, da v povprečju oglase preglejujejo več kot petkrat na dan.

1.4 Cilj diplomske naloge

Cilj diplomske naloge je bil izdelati celovito rešitev, s katero bo uporabniku olajšano nakupovanje rabljenih predmetov na spletu. Rešitev vsebuje spletno aplikacijo, mobilno aplikacijo ter razširitev za brskalnik Google Chrome. Za enostavno povezovanje omenjenih odjemalskih delov rešitve ter dopuščanja odprtih možnosti pri nadgrajevanju, je osrednji del aplikacije zgrajen iz mikrostoritev, kar poleg omenjenih prednosti omogoča tudi skaliranje različno obremenjenih delov infrasrukture. Težnja je predvsem dodobra razviti zaledni del aplikacije, ne pa tudi vseh odjemalskih sistemov. Pri teh smo se osredotočili predvsem na funkcionalnost in ne na estetiko, saj smo želeli bolj kot uporabniški vmesnik predstaviti sobivanje vseh teh sistemov celovite rešitve.

Poglavje 2

Pregled in opis uporabljenih tehnologij

V tem poglavju predstavljamo tehnologije, ki smo jih uporabili pri izgradnji vseh delov celovite rešitve, ter argumentiramo, zakaj smo se odločili za tak pristop. Bolj tehnično orientirano je poglavje o razvoju rešitve, ki sledi. Rešitev, ki smo si jo zadali, je sestavljena iz večih komponent in sicer:

- zaledni oziroma jedrni del aplikacije,
- spletna aplikacija,
- mobilna aplikacija in
- razširitev v spletnem brskalniku.

Pri implementaciji komponent smo imeli na voljo različne tehnologije, med katerimi smo izbrali najprimernejše za našo rešitev, in so tudi predstavljene v nadaljevanju.

2.1 Java, JavaEE in aplikacijski strežnik Wildfly

Java je objektno orientiran programski jezik, ki je bila kot različica 1.0 objavljena leta 1995 [17], v času pisanja te diplomske naloge pa je najnovejša stabilna verzija osnovne Jave (JavaSE - Standard Edition) Java SE 8. Vzdržuje jo Sun Microsystems, ki ga je leta 2010 kupil Oracle Corporation [30].

Poleg osnovne verzije poznamo tudi JavaEE verzijo oziroma Java Enterprise Edition. Je le razširitev JaveSE, kar pomeni, da omogoča vse funkcionalnosti JaveSE ter dodatne, kot so na primer poizvedovanje za podatki in objekti glede na imena (JNDI), enterprise javanska zrna (EJB), Java Servlets in druge. Za gradnjo spletne aplikacije standardna Java ni dovolj, saj potrebujemo poleg osnovne Jave vsaj še podporo javanskih servletov, ki skrbijo za streženje funkcionalnosti spletne aplikacije.

Glavna prednost Jave, kot so si jo zamislili pri načrtovanju pri Sun Microsystems, je povzeta v besedno zvezo WORA, kar pomeni „Write once, run anywhere“. Temelji na principu, da lahko vsak program, spisan v tem programskem jeziku, teče na katerikoli napravi, ki ima nameščen JVM (ang. Java Virtual Machine). Poleg tega so pomembne prednosti tudi, da je stara več kot 20 let, zato ima posledično veliko podporo programerske skupnosti, razvoj samega jezika pa je predvidljiv in stabilen.

Vse komponente JaveEE tečejo na strežniku, ki je lahko bodisi spletni ali aplikacijski. Medtem ko spletni strežnik skrbi zgolj za streženje statičnih datotek, pa aplikacijski skrbi tudi za vso poslovno logiko in jo navzven izpostavlja prek APIja, ki je v primeru izbrane verzije JavaEE EJB. V konkretnem primeru te diplomske naloge, ki temelji na mikrostoritvah in čim bolj okretnih odjemalskih delih, bi lahko za spletno aplikacijo imeli le spletni strežnik, ki bi do podatkov dostopal prek zalednega dela, izpostavljenega na aplikacijskem strežniku.

Za aplikacijski strežnik smo si izbrali Wildfly, ki je bolj splošno znan pod imenom JBoss, v omenjeno ime Wildfly pa je bil preimenovan z različico

8.0.0 [36]. Je aplikacijski strežnik, ki ga je načrtovala organizacija Red Hat, napisan pa je v programskem jeziku Java in implementira specifikacijo verzije JavaEE. Je tudi odprtokodni, glavni prednosti pred konkurenčnimi stežniki pa sta hitro izvajanje in okretnost pri konfiguraciji.

2.2 Maven

Maven [23] je orodje za avtomatsko gradnjo in pakiranje primarno aplikacij, napisanih v Javi. Glavna prednost Maven orodja, ki je bila izkoriščena v procesu izdelave produkta te diplomske naloge, je da omogoča enostavno dodajanje zunanjih knjižnic, ki same po sebi niso del Jave. Vse knjižnice, ki so na voljo, so zbrane v osrednjem repozitoriju, imenovanem Maven Central. Nasploh Maven skrbi tudi za to, kako je osnovna koda na koncu zgenerirana v želeno stanje (jar, war, ear) ter kaj se z njo dogaja pred in po prevajanju. Omogoča tudi enostavno razširitev z vtičniki, ki so na volju na spletu, lahko pa jih spišemo tudi sami.

V praksi in tematsko omejeno na to diplomsko nalogo, je Maven poenostavil celoten proces izdelave aplikacije vse od programiranja do postavitve aplikacije na nek strežnik.

2.3 JavaEE mikrostoritve - Spark Java

Zaledni del rešitve skrbi za shranjevanje, posodabljanje in streženje podatkov, pridobivanje podatkov iz drugih spletnih portalov ter avtorizacijo in avtentikacijo. Zaradi narave aplikacije je celotni zaledni del razdeljen na več posameznih enot oziroma mikrostoritev (več o tem v naslednjem poglavju). Ker smo želeli obdržati programski jezik Java smo se odločili, da za implementacijo mikrostoritev uporabimo Spark (Spark Java [28]). Privlačno pri tem orodju je predvsem to, da omogoča preprosto vzpostavitev REST končnih točk, kar je ena najpomembnejših funkcionalnosti mikrostoritev (komunikacija med sabo). Orodje je bilo razvito okoli Jave 8, tako da z uporabo

lambda izrazov omogoča hitro in vitko programiranje (Koda 2.1 prikazuje lambda izraz za vzpostavitev API točk). Po prevajanju kode, sprogramirane v tem ogrodju, dobimo paket JAR (ang. Java Archive), ki ga postavimo na poljuben spletni strežnik, Spark pa že vključuje Jetty Servlet Engine [18], ki je namenjen streženju servletov.

Obstaja še nekaj alternativ, ki smo si jih ogledali, ko smo se odločali za orodje za vzpostavitev mikrostoritev v Javi.

Nekatere izmed teh alternativ so:

- KumuluzEE [22],
- Wildfly Swarm [37],
- Dropwizard [8].

KumuluzEE ter Wildfly Swarm smo tudi praktično preizkusili, vendar smo se na koncu zaradi hitrega programiranja v lambda izrazih odločili, da ostanemo na Spark Java. Drugače pa je velika prednost obeh ta, da omogočata modularno izbiranje funkcionalnosti JaveEE. Predvsem zato ne izključujemo možnosti, da bomo v prihodnje preklopili na katerega izmed njiju. Lepota mikrostoritev pa je med drugim tudi ta, da nam ni treba prepisati vseh, temveč popravimo le tisti modul, ki bi potreboval spremembe. V času pisanja te diplomske naloge pa ni predvideno, da bi katerakoli izmed treh mikrostoritev, ki so bile načrtovane, potrebovala JavaEE funkcionalnosti.

Koda 2.1: Lambda izrazi pri vzpostavitvi API točk

```
1 put(ADVERTISEMENTS_UPDATE, (req, resp) -> {
2     Advertisement ad = Router.getGson().fromJson(req.body(), Advertisement.class);
3     if (ad != null) {
4         boolean updated = advertClient.updateAdvertisement(ad);
5         return updated ? HttpStatus.SC_OK : HttpStatus.SC_BAD_REQUEST;
6     } else {
7         return HttpStatus.SC_BAD_REQUEST;
8     }
9 }, json());
```

2.4 PostgreSQL in sql2o

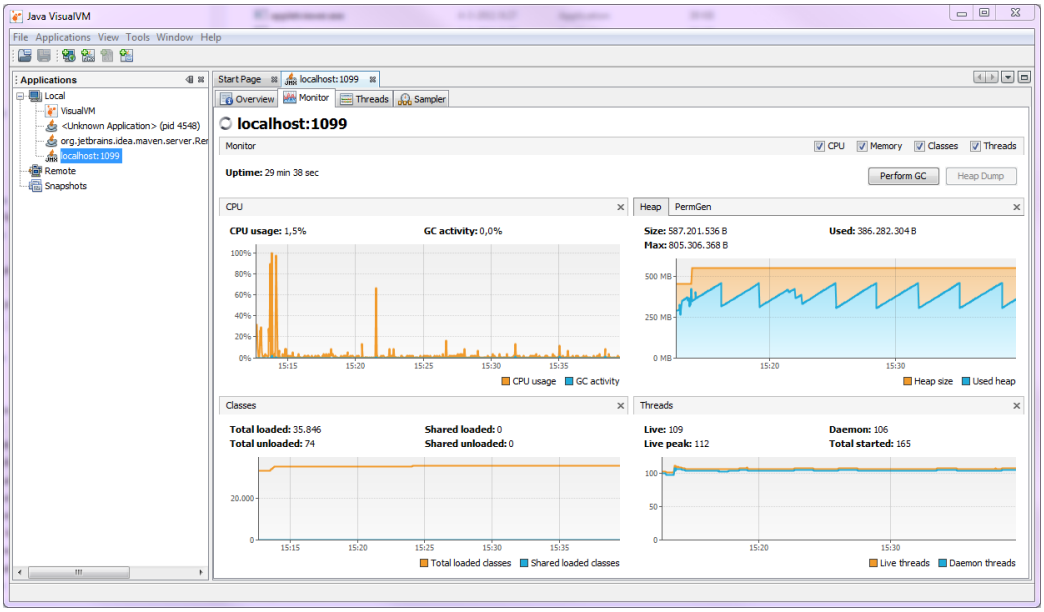
PostgreSQL [25] je objektno orientiran sistem za upravljanje podatkovnih baz in je alternativa MSSQL-u, Oracle-u in drugim. Poglavitna prednost je predvsem hitrost in zanesljivost upravljanja s podatki, poleg tega pa, čeprav je po specifikaciji objektno orientirana, omogoča JSON (ang. JavaScript Object Notation) polja in je tako neke vrste hibrid med objektno in relacijsko bazo. To smo izkoristili v produktu te diplomske naloge in smo vse podatke, nad katerimi niso predvidena iskanja, shranili v JSON formatu.

Sql2o [29] je vitko javansko ogrodje za enostavno izvajanje SQL (ang. Structured Query Language) ukazov in temelji na JDBC (ang. Java Database Connectivity) modulu ter tako deluje na vsaki bazi, ki je skladna z JDBC logiko. Poleg abstrakcije navadnega JDBC ponuja tudi pretvarjanje rezultatov SQL ukaza neposredno v javanski objekt.

Sql2o smo izbrali kot alternativno JPA-ju (ang. Java Persistence API) predvsem zato, ker uporabljamo PostgreSQL kot hibrid med objektno in relacijsko bazo. Medtem kot JPA omogoča ORM (objektno relacijsko mapiranje), imamo pri Sql2o bolj odprte možnosti tako za shranjevanje kot tudi za pridobivanje podatkov iz baze, kar bolj sovpada z želeno hibridno funkcionalnostjo PostgreSQL-a.

2.5 VisualVM

VisualVM [32] je orodje za spremljanje parametrov izvajanja aplikacij, ki tečejo na JVM (ang. Java Virtual Machine). Sprva sicer nismo predvideli potrebe po takšnem orodju, se je pa kasneje pri postavitvi aplikacije pred prve uporabnike in s pojavitvijo prvih problemov pokazalo, da potrebujemo orodje za opazovanje, kako se aplikacija odziva ob različnih obremenitvah. VisualVM omogoča pregled nad osnovnimi parametri kot so obremenjenost procesorja, količina spomina, ki ga aplikacija porabi, in podobno (Slika 2.1). Alternativ je veliko, glavna prednost omenjenga orodja pa je, da je brezplačen in da je namestitev res enostavna.



Slika 2.1: VisualVM pregled parametrov delovanja strežnika

2.6 Google Analytics

Želja po tem, da naredimo aplikacijo uporabniku čim bolj prijazno ter da imamo podatke o obiskanosti in popularnosti naše rešitve, je privedla do tega, da smo uporabili orodje kot je Google Analytics [13]. Orodje omogoča sledenje števila uporabnikov ter interakcij uporabnikov z aplikacijo - torej med drugim tudi to, katere funkcionalnosti so največkrat uporabljene in katere mogoče niti niso potrebne. Uporaba je preprosta, saj zahteva, da v aplikacijo vključimo le nekaj vrstic preddefinirane kode, na voljo pa imamo celoten Googlov portal statističnih podatkov o uporabnikih.

2.7 GIT

Za verzioniranje kode smo uporabili orodje Git [10] (z integracijo na portal GitHub [11]), ki poleg verzioniranja omogoča tudi enostavno sodelovanje z razvijaci. V primeru te diplomske naloge smo ga uporabili predvsem zaradi funkcionalnosti verzioniranja, enostavnega sledenja spremembam ter

preprečitve izgube podatkov. Na GitHub smo tako za vsak modul celovite rešitve ustvarili svoj repozitorij (zaledni del, spletna aplikacija, razširitev v brskalniku in mobilna aplikacija).

2.8 Razširitve v brskalniku Google Chrome

Del ponujene rešitve je tudi razširitev v brskalniku Google Chrome. Razširitve brskalnika so samostojne aplikacije, ki tečejo v sožitju z brskalnikom samim. Razvijalcem nam omogočajo, da smo del uporabnikovega brskanja po spletu. V primeru celovite rešitve, ki jo ponujamo, je to predvsem del, ko shranimo, kaj si uporabnik želi na spletu kupiti, in to uporabimo v drugih delih naše rešitve. Tehnično gledalo so aplikacije napisane v odjemalskih programskih jezikih kot je JavaScript, poleg tega pa se uporablja tudi HTML in CSS.

Poglavje 3

Razvoj rešitve

Pred razvojem aplikacije je treba narediti načrt in opredeliti posamezne funkcionalnosti ter na podlagi tega modularno pristopiti k problemu. Pri našem problemu vemo, da bo celovita rešitev imela več odjemalskih modulov, ki bodo dostopali prek ene vhodne točke v zaledni sistem. Tako pri odjemalskem delu ne pride do težav, saj nadaljnja modulacija ni potrebna (en odjemalec je v tem smislu en modul), pri zalednem delu pa je nujen premislek, saj moramo predvideti, kakšne vse funkcije bo zaledni sistem izvajal, in ga ustrezno razbiti na več mikrostoritev.

V prvem podpoglavju tega poglavja se tako osredotočimo na ta arhitekturni problem celovite rešitve ter na kratko tudi prikažemo kakšen je podatkovni model, nato pa se v naslednjih podpoglavjih poglobimo v zaledni del ter v vsak odjemalski del posebej.

3.1 Načrt razvoja aplikacije

Zaradi različno zahtevnih funkcionalnosti na zalednem sistemu smo se odločili, da bomo k problemu pristopli z mikrostoritvami, ki poleg ostalih prednosti, k našemu problemu pripomorejo predvsem z možnostjo horizontalnega skaliranja posameznih modulov. Glede na naloge zalednega sistema smo te mikrostoritve razdelili na:

- mikrostoritev za avtentikacijo in avtorizacijo,
- mikrostoritev za pridobivanje podatkov z ostalih spletnih strani (ang. web scraping),
- mikrostoritev za manipulacijo podatkov in delovanje algoritma.

V mislih imamo predvsem potencialen problem obremenjenosti modula za pridobivanje podatkov z ostalih spletnih mest, za katerega načrtujemo, da bo najbolj obremenjen. Prav tako je za ta problem najbolj dovzeten tudi zadnje omenjen modul, ki bo poleg naštetih funkcionalnosti imel tudi vlogo vhodne točke do vseh drugih mikrostoritev. Samo obremenjenost bomo spremljali in merili s prej omenjenim orodjem VisualVM.

3.1.1 Arhitektura

Glede na opisane probleme in želje smo si zamislili arhitekturni sistem, ki je prikazan na sliki 3.1. Kot smo že večkrat omenili, naša celovita rešitev predvideva več odjemalskih modulov:

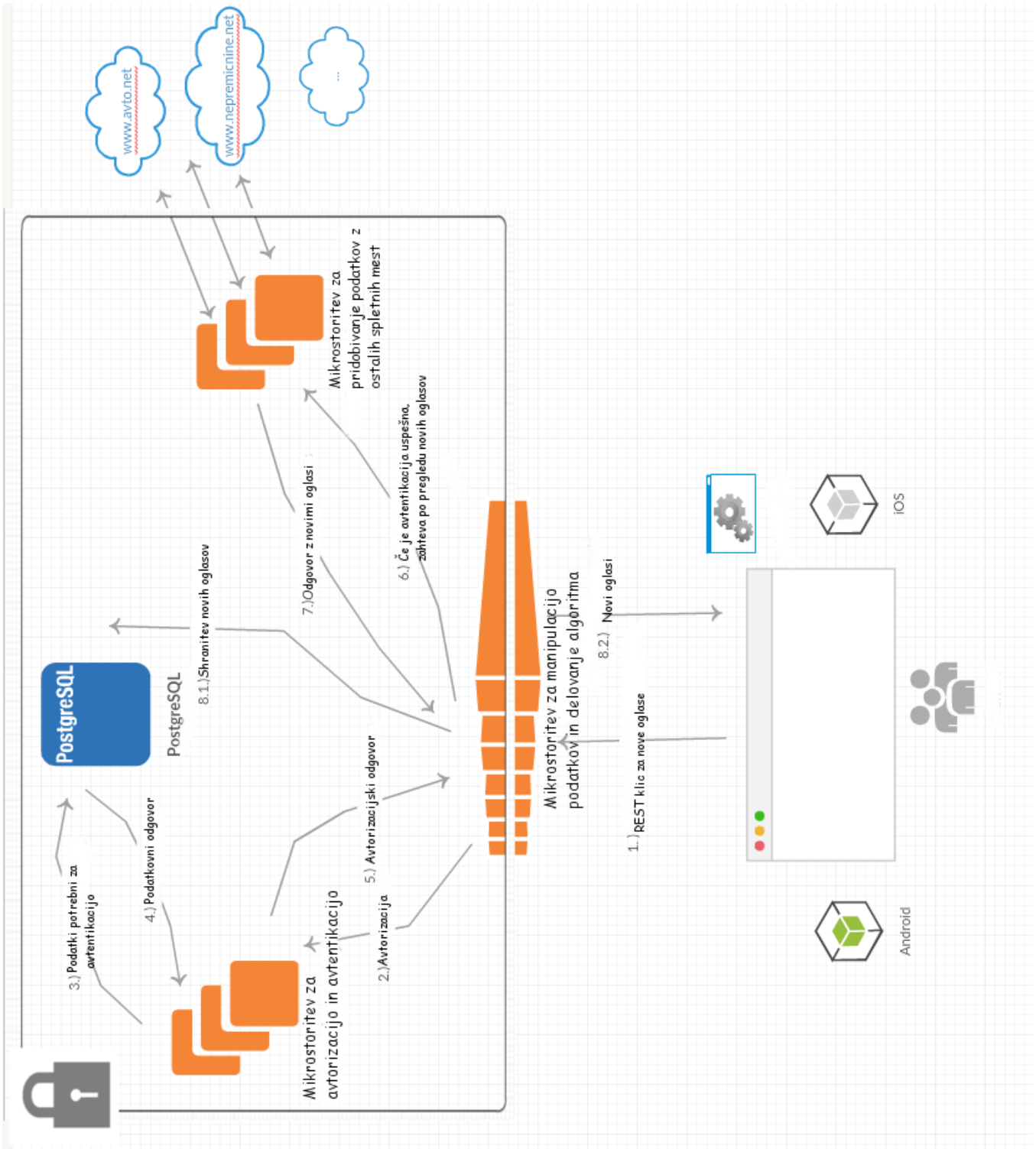
- mobilno aplikacijo,
- spletno aplikacijo in
- razširitev za brskalnik Google Chrome.

Vsi ti odjemalski moduli bodo do vseh podatkov, ki jih potrebujejo za izvajanje, dostopali z REST klici (ang. representational state transfer) na zaledni sistem prek enotne vhodne točke (na sliki 3.1 poimenovana mikrostoritev za manipulacijo podatkov in delovanje algoritma). Ta vhodna točka bo edini del zalednega sistema, ki bo dostopen navzven. Tako baza kot druge mikrostoritve bodo torej praktično nedostopne zunanjemu svetu, s čimer vzpostavimo nek enoten varnostni sistem. Poleg varnosti pa zagotovimo tudi, da nobenemu izmed odjemalskih modulov ni treba vedeti, kaj se dogaja v zalednem delu - kakšne mikrostoritve skrbijo za pridobivanje podatkov, kaj se pred tem

dogaja in koliko mikrostoritev teče v ozadju. Zavedajo se le te vhodne točke in kakšne parametre morajo posredovati za uspešno izveden klic.

Vsak klic na zaledni sistem in tako vsako zahtevo po kakršnih koli podatkih bo vhodna točka najprej poskušala avtorizirati na mikrostoritvi, ki skrbi za varnost (na sliki 3.1 poimenovana mikrostoritev za avtentikacijo in avtorizacijo), nato pa bo le v primeru pozitivnega odgovora te mikrostoritve nadaljevala s prvotno poizvedbo. Primer pridobivanja novih oglasov je po korakih narisan na sliki 3.1. Najprej odjemalski del (npr. mobilna aplikacija) pošlje zahtevek na vhodno točko v zaledni sistem. Zahtevek mora vsebovati avtentikacijski žeton, ki ga uporabnik dobi ob prijavi (ali ob osvežitvi prvotnega žetona) in parametre poizvedbe. Avtentikacijski žeton vhodna točka preusmeri na mikrostoritev, ki skrbi za varnost in uporabnika poskusi avtenticirati. Šele po uspešni avtentikaciji se vhodna točka začne ukvarjati s prvotno poizvedbo. Glede na parametre se ustvari zahtevek za mikrostoritev, ki skrbi za pridobivanje podatkov z ostalih spletnih strani. Ta izvede to zahtevo, pretvori podatke v javanski objekt, ki ga vse mikrostoritve poznajo, ter to vrne nazaj na vhodno točko. Vhodna točka na tej stopnji vrne vse nove oglase na odjemalski del, ki je zahteval poizvedbo, asinhrono pa te oglase shrani tudi v podatkovno bazo.

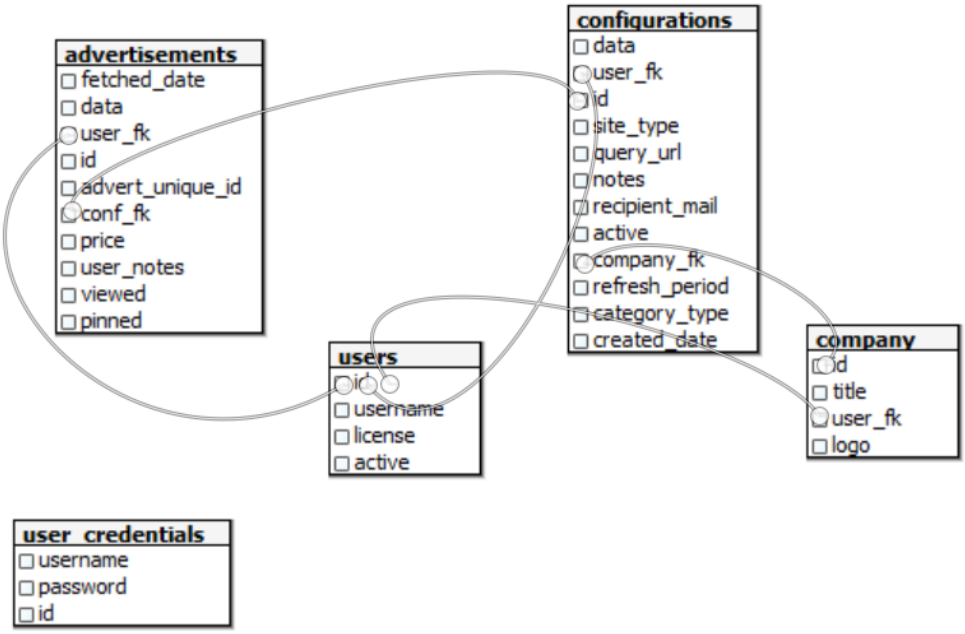
Glede na obremenjenost posameznih mikrostoritev jih lahko poljubno horizontalno skaliramo, pri čimer je treba poudariti, da v tem primeru potrebujemo sistem izbiranja na katero instanco bomo posredovali klic. To potrebujemo tudi v primeru, če katera izmed instanc mikrostoritev odpove. Ta del rešitve sicer ni implementiran v sklopu te diplomske naloge, smo si pa ogledali tudi nekaj potencialnih rešitev za ta problem, ena izmed katerih je implementacija Apache Camela [3].



Slika 3.1: Arhitektura rešitve

3.1.2 Podatkovni model

Kot narekuje teorija smo si osnovni podatkovni model osmislili že pred samim začetkom programiranja. Da je celotno načrtovanje agilnejše, pa se je izkazalo, ko se je podatkovni model začel korenito spreminjati pri samem razvoju aplikacije. Ob pisanju te naloge in zaključku trenutne stopnje aplikacije je podatkovni model sestavljen iz petih tabel, ki so prikazane na sliki 3.2. Na tem mestu bi opomnili, da uporabljamo hibrid PostgreSQL med relacijsko in objektno usmerjeno podatkovno bazo tako, da so vsi podatki, po katerih ni predvideno iskanje, vpisani v JSON obliki. Tak primer je npr. v tabeli „advertisements“ polje „data“.



Slika 3.2: Podatkovni model, uporabljen pri rešitvi v okviru diplome. Sestavljen je iz petih tabel, relacije med posameznimi tabelami pa so predstavljene s povezavami

Users

V tabeli „uporabniki“ (ang. users) so shranjeni ključni podatki o uporabniku, ki se registrira v aplikacijo. Tako tabela hrani le polja kot so uporabniško ime, licenca, podatek o tem ali je aktiven ali ne ter edinstven identifikator. Zaradi varnosti podatkov ni neposredne relacije med to tabelo ter tabelo „user credentials“, ki hrani gesla.

User credentials

V tabeli „user credentials“ je shranjeno zgoščeno geslo z algoritmom SHA-512 pod referenco uporabniškega imena, ki je prav tako zgoščeno s to funkcijo in drugačnim podpisom (Koda 3.1). S takšnim pristopom praktično ni možnosti, da bi potencialni napadalec sistema prišel do povezave uporabnika in njegovega gesla.

Koda 3.1: Zgoščevanje gesla in reference na uporabnika

```
1  public static String digest(String string) {
2      return digest(DIGEST_SIGNATURE_OUTER,
3          digest(string, DIGEST_SIGNATURE_INNER, 10, 70), 20, 50);
4  }
5
6  private static String digest(String string, String key, int start, int end) {
7      return StringUtils.substring(
8          Base64.encodeBase64String(DigestUtils.sha512(string + key)),
9              start, end);
10 }
```

Advertisements

V tabeli „oglasi“ (ang. advertisements) so shranjeni vsi podatki o oglasih, referenca na uporabnika ter referenca na njegovo konfiguracijo. Podatki, ki jih dobimo z ostalih spretnih portalov, so v JSON obliki shranjeni v polje „data“, drugi podatki, po katerih predvidevamo tudi iskanje, pa so v posameznih stolpcih.

Configurations

Vsak uporabnik ima tudi eno ali več konfiguracij shranjenih v tabeli konfiguracije (ang. configurations), ki vsebuje vse podatke o želenem iskanju, ki mu sledi. Tako na primer hranijo tip strani, naslov do strani, čas osveževanja oglasov, datum nastanka konfiguracije, e-poštni naslov za prejemanje obvestil in druge podatke.

Company

Tabela „podjetje“ (ang. company) je entiteta, na katero je vezana posamezna konfiguracija. Podjetje ima lahko nase vezano eno ali več konfiguracij, vsebuje pa osnovne podatke o podjetju, ki je oglas objavilo. Če je to podjetje tudi član našega portala, vsebuje še referenco s ključem na uporabnika.

3.2 Zaledni del

Zaledni del celovite rešitve je sestavljen iz vhodne točke, ki je sama po sebi mikrostoritev, in ostalih mikrostoritev, ki smo jih modularno razdelili glede na funkcionalnosti, za katere želimo, da jih izvajajo. Med seboj komunicirajo prek REST protokola, za vse pa velja razdelitev obveznosti (ang. separation of concern). Nobeni torej ni potrebno vedeti, kako svoje delo opravlja druga mikrostoritev. Zavedajo se le funkcionalnosti in načina, kako jo morajo uporabljati (kakšne parametre morajo posredovati za želen odgovor).

3.2.1 Mikrostoritev za manipulacijo podatkov in delovanje algoritma

To je mikrostoritev, ki je hkrati tudi vhodna točka v celotni zaledni sistem in poleg preusmerjanja klicev glede na vhodne parametre skrbi tudi za pridobivanje, posodabljanje in shranjevanje podatkov. Vsak klic na zaledni sistem gre od odjemalskih modulov skozi to točko, ki najprej poskrbi, da je zahtevek avtenticiran in nato izvede vso logiko glede na zahtevek (Koda 3.2).

Koda 3.2: Prestreženje vsakega zahtevka

```
before((request, response) -> {
    String token = request.headers(HttpHeaders.AUTHORIZATION);
    if (!requestIsExcused(request) &&
        StringUtils.isEmpty(Router.authToken(token))) {
        halt(401);
    }
});
```

Poleg osnovnih nalog, kot so pridobivanje podatkov iz podatkovne baze, posredovanje pri pridobivanju novih oglasov z drugih spletnih strani ter posredovanje pri zahtevkih za prijavo, skrbi tudi za delovanje obveščevalnega algoritma. To je algoritem, ki glede na uporabnikove zahteve poskrbi za obveščanje o novih oglasih. Po uporabnikovi zahtevi, da želi biti na neko časovno periodo obveščen o novih oglasih, se na tej mikrostoritvi ustvari časovni načrt za vsako izmed konfiguracij (oziroma so za zmanjšanje prometa med mikrostoritvami te konfiguracije razdeljene v skupine glede na isto spletno stran in isto časovno enoto). Ti časovni načrti se izvajajo z uporabo knjižnice Quartz-Scheduler [26], ki omogoča tudi, da se določena metoda v Javi izvede na neko časovno enoto (Koda 3.3).

Ko torej pride do potrebe po pregledu novih oglasov, se najprej iz baze prebere vse konfiguracije uporabnika, združi v skupine glede na spletno mesto za zmanjšanje števila klicev in pokliče mikrostoritev za pridobivanje podatkov z ostalih spletnih mest. Ta nato vrne vse oglase, ki jih najde, algoritem pa na tem mestu iz baze prebere vse trenutne oglase, ki spadajo k tej konfiguraciji, jih primerja in izlušči nove. Nato algoritem zgenerira elektronsko pošto z osnovnimi podatki o teh oglasih ter ga pošlje na uporabnikov e-poštni naslov (Slika 3.3 prikazuje primer takšnega sporočila). Za prihodnje takšne klice mora na tem mestu te nove oglase algoritem tudi shraniti v podatkovno bazo.

Koda 3.3: Uporaba knjižnice Quartz-Scheduler za izvajanje časovnih načrtov

```
private void scheduleUserTimer(Scheduler scheduler, Configuration... configuration)
        throws SchedulerException {
    if (configuration == null || configuration.length < 1) {
        return;
    }
    String identities = constructIdentities(configuration);
    JobDetail job = newJob(NightlyWorker.class)
        .withIdentity(identities, identities)
```

```
9           .build();
10
11      int minutes = configuration[0].getRefresh_period();
12
13      Trigger trigger = newTrigger()
14          .withIdentity(identities, SCRAPER_TRIGGER)
15          .startNow()
16          .withSchedule(simpleSchedule()
17              .withIntervalInMinutes(minutes)
18              .repeatForever())
19          .build();
20
21      // Tell quartz to schedule the job using our trigger
22      scheduler.scheduleJob(job, trigger);
23  }
```

Slika 3.3: Izgled e-poštnega sporočila, ki se pošlje v primeru novih oglasov

3.2.2 Mikrostoritev za avtentikacijo in avtorizacijo

Ta mikrostoritev je popolnoma avtonomna in ima svoj dostop do baze podatkov, s čimer dopuščamo možnost, da jo v prihodnje brez težav zamenjamo

s kakšno izmed že obstoječih rešitev za avtentikacijo in avtorizacijo (npr. Keycloak [21]).

Naloge mikrostoritve so avtentikacija, avtorizacija in generiranje podatkov potrebnih ob registraciji novega uporabnika. Ta celoten del je izoliran zgolj na to mikrostoritev in je tako edini del arhitekture, ki zna iz avtentikacijskega žetona pridobiti in avtenticirati uporabnika. Takšna rešitev lepo uprizarja razdelitev obveznosti (ang. separation of concern), na podlagi katere smo poskušali ločiti vsako izmed mikrostoritev.

Registracija deluje tako, da se kot vhodni parameter v mikrostoritev pošlje zakodirano uporabniško ime in geslo po Base64 sistemu kodiranja. Po dekodiranju se iz uporabniškega imena s SHA-512 kriptirnim algoritmom ustvari referenca na uporabnika. Zgoščevanje je potrebno zato, da potencialni napadalec v sistem ne more glede na uporabniško ime najti gesla. V primeru, da ima uporabniško ime in pride do tabele, kjer so shranjeni ti podatki, so vse reference na uporabnika zgoščene in ker ne pozna podpisa, ki je bil uporabljen za zgoščevanje, ne more najti ujemajočega se gesla.

Z enakim kriptirnim algoritmom SHA-512 je zgoščeno tudi želeno geslo, kar prepreči, da bi kdorkoli iz vnosov v tabeli izvedel prvotno nezgoščeno geslo. Kriptirni algoritem SHA-512 namreč ne omogoča dekodiranja v prvotno obliko.

Podobno deluje tudi avtentikacija, ki se izvede ob vsakem klicu v zaledni sistem. Vhodni podatek je v tem primeru žeton, ki je dodeljen uporabniku ob prijavi v sistem. Ta žeton (HMAC, ang. hash message authentication code) se zgenerira iz uporabniškega imena, ki se ob zahtevi izlušči iz žetona in uporabi pri avtentikaciji (Koda 3.4). Pri tem procesu je dovolj zgolj to, da preverimo, če obstaja vnos s to referenco uporabniškega imena.

Koda 3.4: Avtentikacija uporabnika pred vsako obravnavo zahtevka

```
1 private static String bearerAuthentication(String auth) {
2     Jwt jwt = proccessBearerToken(auth);
3     if (jwt != null) {
4         String username = ((DefaultClaims) jwt.getBody()).getSubject();
5         UserCredentials userCredentials = Router.getUserCredentials(digest(username));
6         if (userCredentials != null) {
7             return createToken(username);
8         }
```

```
9       }
10      return null;
11  }
```

3.2.3 Mikrostoritev za pridobivanje podatkov z drugih spletnih strani

Ta mikrostoritev skrbi za pridobivanje podatkov z drugih preddefiniranih spletnih strani. Navzven odpira več vhodnih točk (API), vsako za svoj spletni portal, s katerim imamo implementirano integracijo za pridobivanje podatkov (Koda 3.5). Tako se že z ločitvijo vhodnih točk na portale rešimo enega potrebnega vhodnega parametra. Glede na to, s katere strani moramo pridobiti podatke, namreč ločimo logiko, kako se to pridobi (vsaka spletna stran ima svojo strukturo). Potreben vhodni parameter je tako le URL naslov, ki ga je uporabnik vnesel kot naslov, kateremu želi slediti. Portali, ki jih lahko podpremo, morajo torej imeti vse kriterije iskanja navedene v URLju, saj prek tega URLja dostopamo konkretno do oglasov, ki jim naš uporabnik sledi (Slika 3.4 prikazuje primer takega URLja).

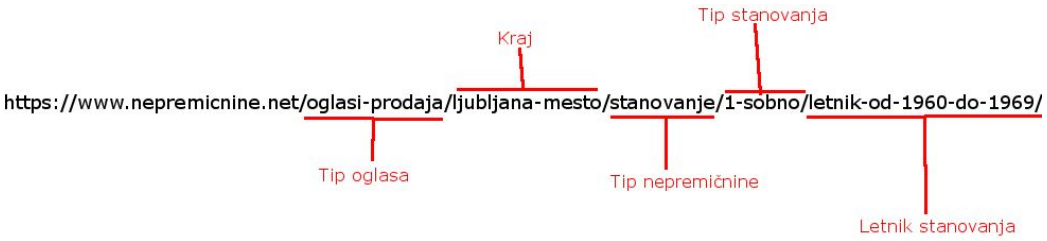
Ko imamo URL, se na to spletno stran povežemo s knjižnico Jsoup [19], ki je javanska implementacija orodja, s katerim lahko iz HTML datoteke pridobimo podatke. Pred tem seveda preberemo robots.txt datoteko, v kateri preverimo kako in če sploh lahko s te spletne strani pridobivamo podatke.

Želene podatke sproti pretvarjamo v javanske objekte, ki jih nato ta mikrostoritev pošlje nazaj kot odgovor na zahtevo.

Koda 3.5: Vsaka API točka za svoj portal

```
1  get(Scraper.MOBILEDE, (req, resp) -> {
2      try {
3          List<AdvertEntity> newAds = new ArrayList<>();
4          newAds = doFetch(toFetch[0], SiteEnum.MOBILEDE);
5          fetchingSite[0] = SiteEnum.MOBILEDE;
6          fetched[0] = newAds.size();
7          return newAds;
8      } catch (IOException e) {
9          LOGGER.error("Error fetching Mobilede: {}", e.getMessage(), e);
10         resp.status(400);
11         return "";
12     }
13 }, json());
14
```

```
15 get(Scraper.AUTOSCOUTIT, (req, resp) -> {
16     try {
17         List<AdvertEntity> newAds = new ArrayList<>();
18         newAds = doFetch(toFetch[0], SiteEnum.AUTOSCOUT_IT);
19         fetchingSite[0] = SiteEnum.AUTOSCOUT_IT;
20         fetched[0] = newAds.size();
21         return newAds;
22     } catch (IOException e) {
23         LOGGER.error("Error fetching AutoscoutIt: {}", e.getMessage(), e);
24         resp.status(400);
25         return "";
26     }
27 }, json());
```



Slika 3.4: Primer URL naslova, iz katerega izločimo iskalne parametre

3.3 Spletna aplikacija

Namen spletne aplikacije je, da ima uporabnik pregled nad vsemi spletnimi oglasi ter da lahko z različnimi moduli, ki jih v aplikaciji ponujamo, pridobi karseda veliko informacij o želenem izdelku - bodisi je to prevozno sredstvo, nepremičnina ali kak drug predmet. Po prejetem elektronskem sporočilu o novem oglasu je uporabnik preusmerjen na spletno aplikacijo, kjer lahko s tem oglasom poljubno upravlja - ga doda med priljubljene, označi kot ogledanega, deli z drugimi, mu doda javen komentar ali zasebni zapisek, ali pa ga enostavno izbriše.

Tehnologija, ki je bila uporabljena pri izdelavi spletne aplikacije, je JSF [20] (ang. Java Server Faces) kot čelni del za prikaz in JavaEE kot del, kjer se izvaja poslovna logika. Grajena je na principu MVC (ang. model view controller), kjer EJBji predstavljajo model spletne aplikacije in podatke pridobivajo z mikrostoritve za manipulacijo podatkov in delovanje algoritma (slika 3.1). Te podatke nato obdelajo in jih ustrezno prikažejo.

3.3.1 Izgled

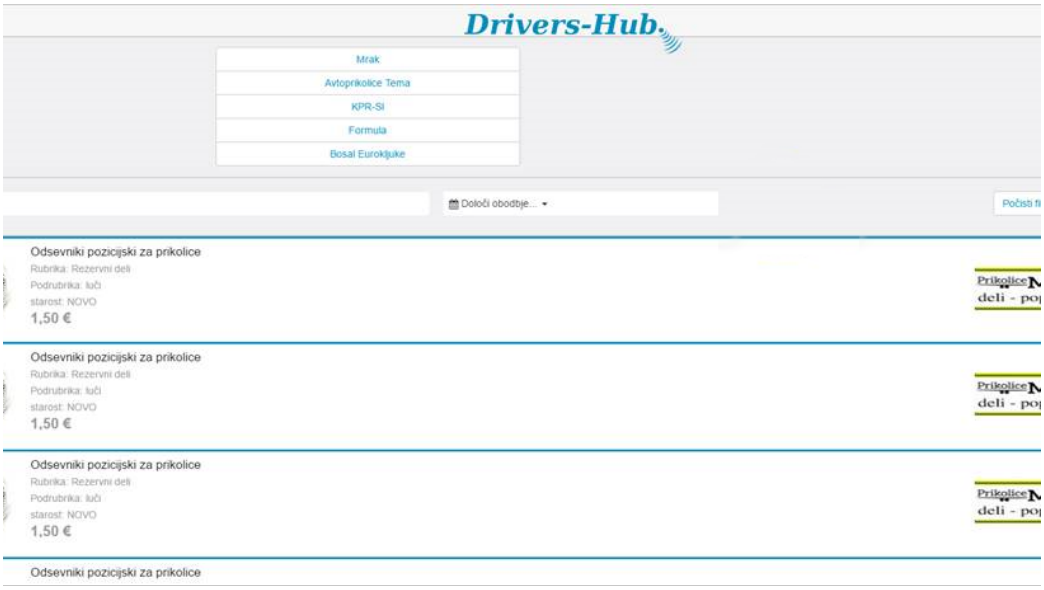
Osnovna knjižnica za izgled, na kateri se je nato gradilo in jo nadgrajevalo, je Bootcards [6]. Bootcards je knjižnjica izdelana na osnovi Bootstrapa v3 [7], zato je prilagojena tako za namizne oziroma prenosna računalnike kot tudi za mobilne in tablične naprave. Temelji na principu kartic (ang. cards), kjer je vsaka funkcionalnost ločena, podatki pa so agregirani in prikazani na eni samostojni kartici [35]. Tako imamo v naši aplikaciji ločene kartice za prikaz vseh oglasov; pri prikazu posameznega oglasa pa kartico za prikaz osnovnih podatkov, kartico za slike, kartico, kjer so prikazane informacije o oglasu s tujih spletnih strani in kartico za komentarje oziroma zapiske.

Naredili smo tudi dva logotipa aplikacije - en manjši, ki je viden v zavihku v brskalniku (Slika 3.5), drug pa večji in združuje ime aplikacije „Adverts-Tracker“ ter prvotni manjši logotip (Slika 3.6).

Izgled aplikacije se je razvijal skupaj z dodatnimi funkcionalnostmi, ki so bile dodane med pisanjem diplomske naloge. Delna evolucija izgleda je vidna na slikah 3.7 in 3.8.

Slika 3.5: Manjši logotip

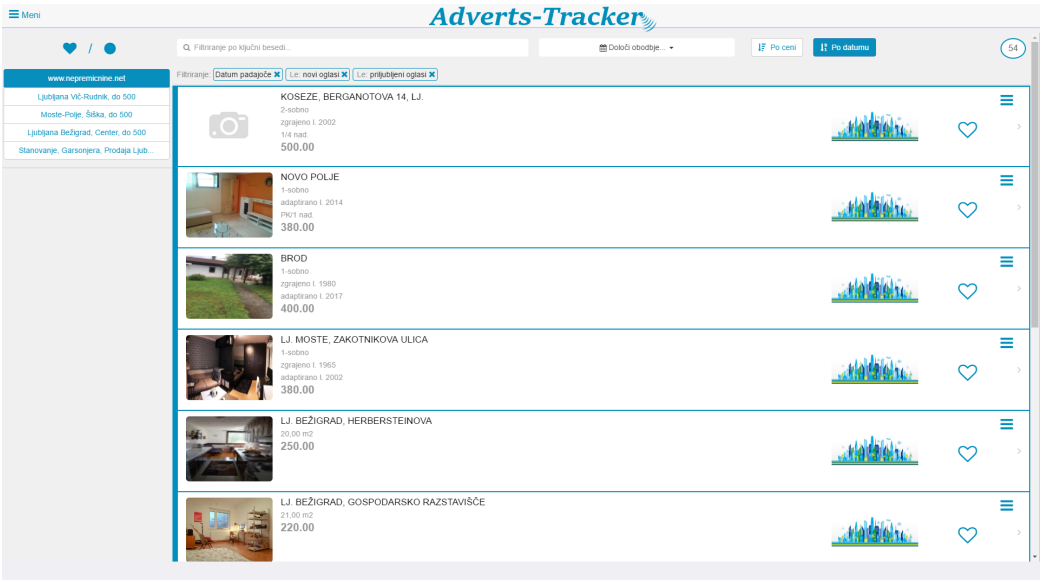
Slika 3.6: Večji logotip



Slika 3.7: Prva verzija izgleda spletne aplikacije

3.3.2 Seznam vseh oglasov

Glavni pogled aplikacije, na katerega je uporabnik preusmerjen po prijavi, je osnovni pregled vseh oglasov z možnostjo naprednega filtriranja (Slika 3.8). Tukaj so prikazani vsi oglasi, ki jih je algoritem dodal v seznam glede na želene kriterije sledenja. Aplikacija nam ponuja več različnih možnosti



Slika 3.8: Trenutna verzija izgleda spletne aplikacije

filtriranja in iskanja po celotni bazi oglasov:

- glede na konfiguracijo, h kateri spadajo,
- glede na spletno mesto, kjer so bili najdeni,
- glede na datum objave,
- glede na to, ali oglas (ali komentar ali zapisek) vsebuje ključno besedo,
- glede na to, ali je oglas priljubljen,
- glede na to, ali je oglas nov.

Vse te kriterije je možno tudi kombinirati, torej imamo možnost iskanja na primer po ključni besedi, v nekem datumskem razponu in samo za točno določeno konfiguracijo. Poleg tega imamo tudi dve možnosti sortiranja: po datumu ali po ceni.

Osnovni pregled nam omogoča tudi enostavno in hitro dodajanje oglasa med priljubljene, prav tako pa imamo na voljo tudi meni z naprednimi

možnostmi, kjer so možnosti kot je deljenje, označevanje oglasa kot prebranega/neprebranega in brisanje.

Oglasi so v tem pogledu privzeto prikazani le v skrajšani obliki - vidimo naslov oglasa, ceno in pa največ tri dodatne podatke. Poleg tega vidimo tudi skrajšano obliko lastnih zapiskov (predolgi zapiski se avtomatsko skrajšajo in končajo s tremi pikami). Ta pogled imenujemo enojni pogled, ki ga s klikom na želen oglas razširimo v dvojni pogled - tako je implementiran pogled glavno-podrobno (ang. master-details).

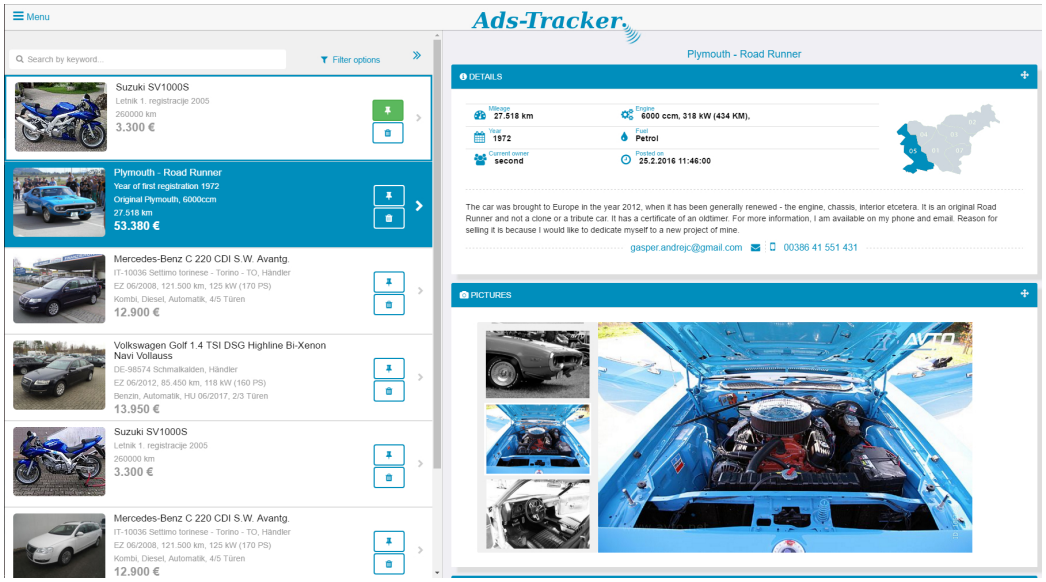
3.3.3 Predogled posameznega oglasa

V dvojnem pogledu se prikažejo vsi podatki o oglasu agregirani na posamezne enote (kartice). Tako imamo posebej na kartici prikazane vse podatke o oglasu, nato vse slike, komentarje, lastne zapiske in podobno (Slika 3.9). Te kartice si lahko uporabnik sam enostavno uredi v želeni vrstni red tako, da prime in z miško postavi kartico na novo pozicijo. Pozicije kartic se v podatkovni bazi shranijo vsakemu uporabniku posebej, s čimer omogočimo, da želeni vrstni red ostane vse do ponovne rekonfiguracije.

Ker želimo ohraniti pridobivanje podatkov s spletnih strani na drugih strežnikih na minimumu, prikazujemo le osnovne podatke. Za vse dodatne informacije o oglasu morajo uporabniki naše aplikacije klikniti na naslov oglasa in si ga ogledati na prvotni spletni strani. Namen naše aplikacije namreč ni, da nadomestimo vse ostale, s katerih pridobivamo podatke, ampak je obveščati uporabnike o novih oglasih in jim omogočiti osnovni pregled, glavni vir informacij pa je prvotna spletna stran.

3.3.4 Komentiranje in zapiski

V dvojnem pogledu oglasa je prisotna tudi kartica za komentiranje oziroma kartica z zapiski. Kartica za komentiranje omogoča vsem uporabnikom, da podajo svoje mnenje o oglasu. Konkreten primer uporabnosti tega je pri nakupu rabljenega vozila, kjer lahko uporabniki, ki so si vozilo že ogledali v



Slika 3.9: Dvojni pogled spletne aplikacije s podrobnim pregledom izbranega oglasa

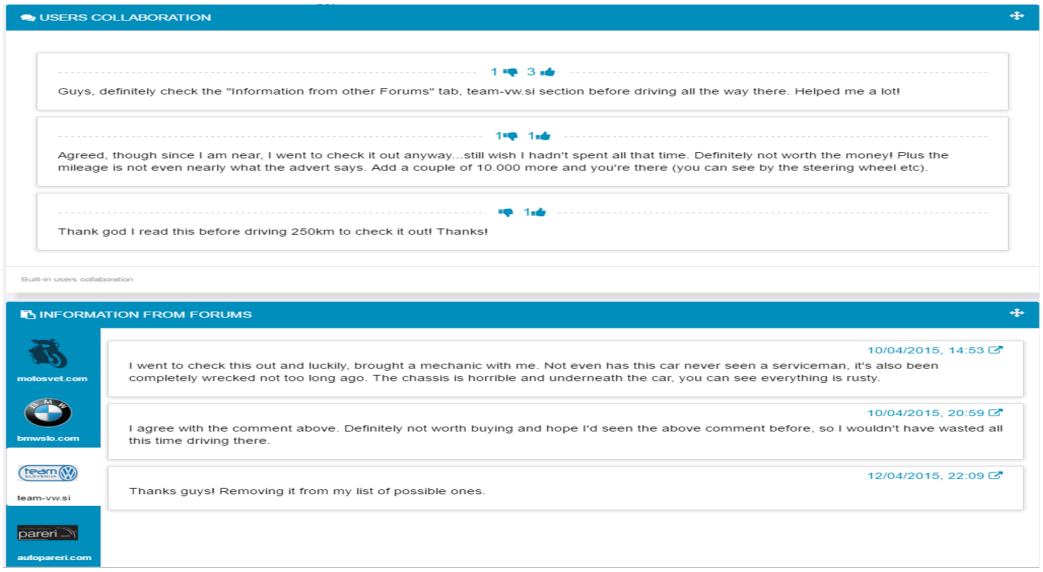
živo, delijo svoje mnenje z ostalimi uporabniki aplikacije in jim tako prihranijo čas, ki bi ga sicer sami porabili za ogled.

Kartica z zapiski ima podobno funkcionalnost, le da so v tem primeru vsi komentarji zasebni in so vidni le avtorju.

3.3.5 Pregled informacij o oglasu na internetu

Kartica s pregledom informacij o oglasu na internetu združuje vse podatke, ki jih najdemo na preddefiniraih spletnih mestih, kot so različni forumi (Slika 3.10). Pri pregledu ali rešitev, ki jo ponujamo z našo aplikacijo, že obstaja, smo naleteli na uporabnost forumov pri prodaji oziroma nakupu rabljenih vozil. Na vseh forumih, ki smo jih uspeli pregledati, namreč obstaja kategorija, kjer si uporabniki izmenjujejo mnenja o vozilu glede na URL naslov oglasa. Tako smo integrirali pridobivanje informacij s teh preddefiniranih forumov (oziroma specifičnih kategorij s teh forumov), kjer iščemo vse objave, ki bi vsebovale URL naslov oglasa, ki ga uporabnik gleda v dvojnem pogledu. Primer takšnega foruma in objave je prikazan na sliki 3.11. V našo aplikacijo

smo prenesli naslov do te objave, kjer lahko uporabnik v večini primerov vidi, ali je kateri izmed uporabnikov foruma napisal kaj o oglasu.

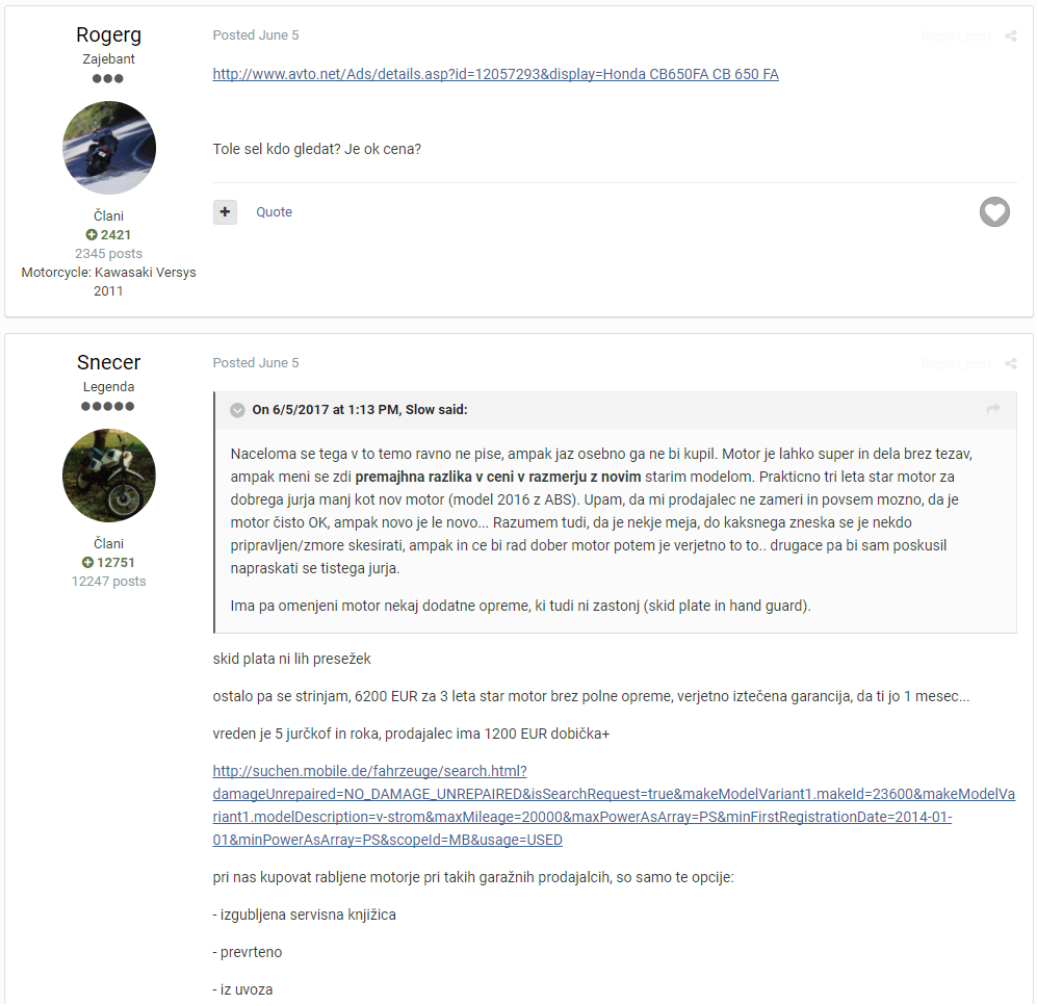


Slika 3.10: Integracija spletne aplikacije z za oglas relevantnimi spletnimi forumi

3.3.6 Nadzorna plošča

Nadzorna plošča je podstran aplikacije, kjer ima uporabnik pregled nad vsemi sledenji in možnost njihovega urejanja. V tabeli, ki je po izgledu znotraj svoje kartice, so izpisani vsi osnovni podatki konfiguracije (prikazano na sliki 3.12):

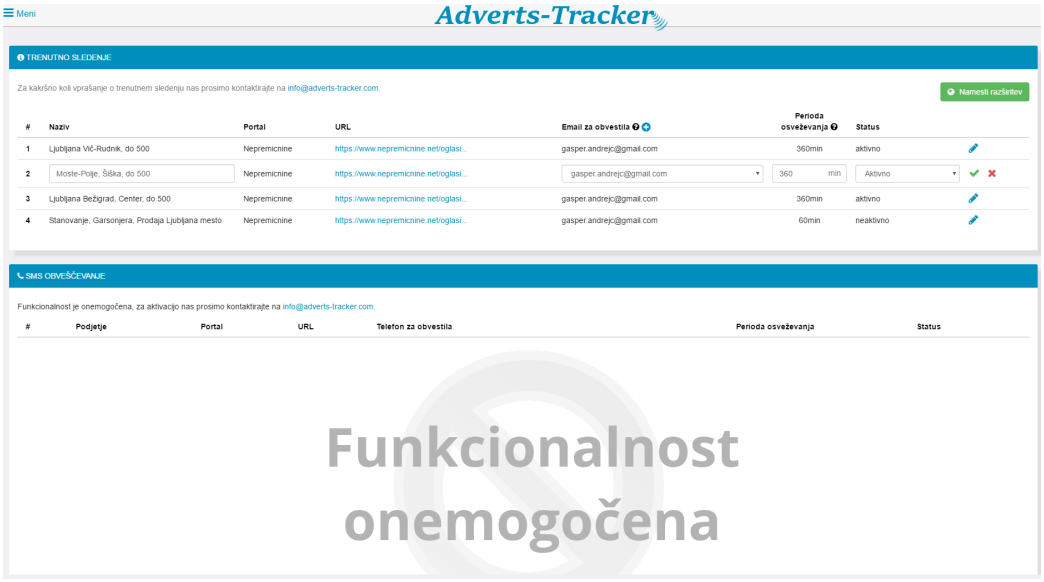
- naziv, ki ga določi uporabnik sam in po katerem prepozna sledenje (npr. enosobno stanovanje, Ljubljana, do 500),
- portal, od koder pridobivamo podatke,
- neposreden URL naslov do rezultatov izbranih kriterijev,
- e-poštni naslov kamor algoritem pošilja obvestila o novih oglasih,
- perioda osveževanja, ki definira na koliko časa algoritem pregleda nove oglase,



Slika 3.11: Primer komentarja na enem izmed spletnih forumov, ko uporabnik vpraša za mnenje o oglasu [24]

- status konfiguracije (aktivno/neaktivno), ki določa, ali algoritem aktivno obvešča o novih oglasih ali ne.

Uporabnik lahko v nadzorni plošči ureja naziv, e-poštni naslov za obvestila, periodo osveževanja in status konfiguracije. Portala in neposrednega URLja uporabnik ne more urejati, saj je portal določen avtomatsko, URL naslov pa je dodan prek razširitve v spletnem brskalniku, saj tako preprečimo potencialno nepravilen vnos (URL naslov je lahko laičnim uporabnikom ne-



Slika 3.12: Nadzorna plošča v spletni aplikaciji

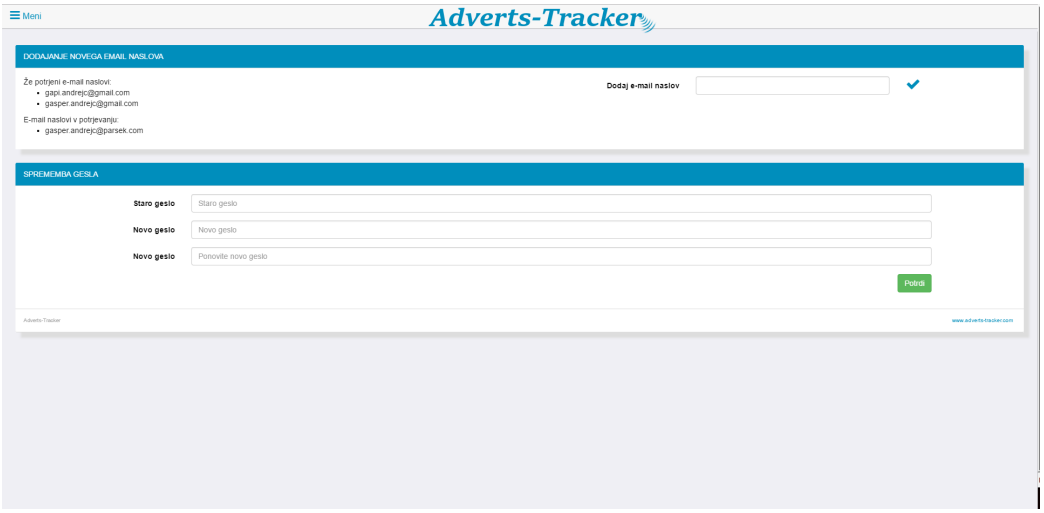
razumljiv).

E-poštni naslov lahko uporabnik spremeni le, če ga je predhodno potrdil. S tem preprečimo, da bi algoritem pošiljal obvestila na napačen ali tuj e-poštni naslov. Izbiranje tega naslova je v uporabniškem vmesniku torej implementirano z izvlečnim seznamom, ki se napolni glede na vse uporabnikove potrjene e-poštne naslove. Ti se dodajajo v nastavitvah.

3.3.7 Nastavitve

Tako kot nadzorna plošča so tudi nastavitve podstran aplikacije (Slika 3.13). Tukaj uporabnik konfigurira svoje geslo, uporabniško ime, privzeto obnašanje aplikacije in dodajanje e-poštnih naslovov, ki jih lahko izbere pri urejanju konfiguracij.

Geslo uporabnik spremeni tako, da vtipka svoje trenutno geslo in dvakrat novo želeno geslo. Če so vsi ti pogoji izpolnjeni, se geslo uporabniku posodobi, sam pa je avtomatsko izpisan iz aplikacije s sporočilom, da je bilo geslo spremenjeno in da je potreben ponovni vpis. V zalednem sistemu se pri tem zgodi podoben proces kot pri registraciji uporabnika - v avtentika-



Slika 3.13: Nastavitve v spletni aplikaciji

cijski glavi (ang. authentication header) se zaradi potrebe po avtorizaciji vsakega zahtevka pošlje trenutno geslo, v posebnem zaglavju (ang. header) imenovanem „ChangedPassword“ pa se pošlje novo geslo. To novo geslo se pošlje na varnostno mikrostoritev, ki zna geslo pravilno zgostiti in pripraviti za shranitev oziroma posodobitev v bazi.

Na tej podstrani lahko uporabnik doda tudi nov e-poštni naslov, ki ga lahko po potrditvi uporabi pri nastavitvah konfiguracij. Vpiše se želeni e-poštni naslov (ki mora biti pravilno formiran, kar se preverja z regularnim izrazom, ki ga prikazuje koda 3.6), na katerega se pošlje potrditveno sporočilo. Pri dodajanju novega e-poštnega naslova se zgenerira unikaten UUID izraz, ki se shrani v podatkovno bazo v zapis za uporabnika in doda kot URL parameter na URL naslov, poslan v potrditvenem e-poštnem sporočilu. Ko uporabnik v e-poštnem sporočilu klikne na ta URL, se v aplikaciji glede na ta UUID poišče uporabnika. Če je iskanje uspešno, vemo, da ima ta uporabnik dostop do e-poštnega nabiralnika, kamor je bilo poslano sporočilo. Tako potrdimo e-poštni naslov in dovolimo, da ga uporabnik uporablja pri obveščanju o novih oglasih.

Koda 3.6: Regularni izraz za preverjanje pravilnosti e-poštnega naslova

```
1 EMAIL_PATTERN = Pattern.compile("^[_A-Za-z\\d\\-\\+]+
2 (\\.[_A-Za-z\\d\\-]+)*@[A-Za-z\\d\\-]+(\\.[A-Za-z\\d\\-]+)*(\\.[A-Za-z]{2,})$");
```

3.4 Mobilna aplikacija

Mobilna aplikacija celovite rešitve lahko nadomesti spletno aplikacijo, a z nekaterimi omejitvami. Prav tako kot spletna aplikacija, tudi mobilna aplikacija omogoča pregled nad vsemi oglasi in enostavno dodajanje med priljubljene oziroma brisanje. V času pisanja te diplomske naloge pa mobilna aplikacija ne omogoča dodajanja lastnih komentarjev, pregleda ostalih informacij s spleta, pregleda drugih komentarjev in deljenja oglasov. Tako je v bistvu okrnjena različica spletne aplikacije, omogoča pa hiter dostop do funkcionalnosti celovite rešitve na mobilnih in tabličnih napravah. Ima tudi svojo funkcionalnost, to je obveščanje s potisnimi sporočili (ang. push notifications) in tako dodaja novo metodo obveščanja (poleg obveščanja o novih oglasih prek elektronskih sporočil).

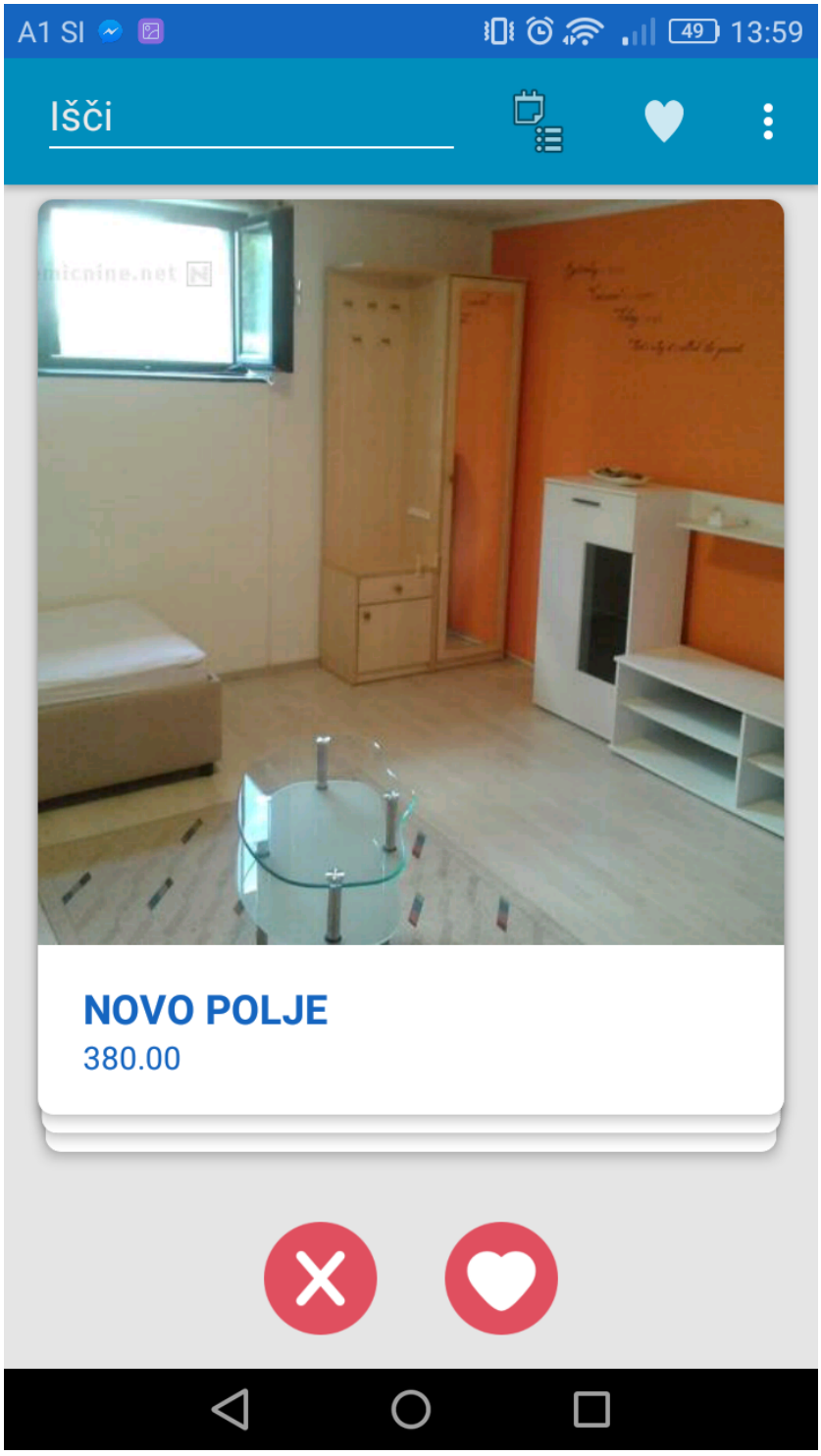
Sprogramirana je v Android Studiu [1] v programskem jeziku Java in je kompatibilna s sistemom Android z imenom JellyBean oziroma starejšimi (uporabljen je SDK verzije 16). Pri programiranju smo uporabili številne knjižnice tako za izgled kot za poslovno logiko.

3.4.1 Izgled in funkcionalnosti

V aplikaciji imamo dva osnovna pogleda - pogled vseh oglasov, kjer imamo možnost tudi osnovnega filtriranja in iskanja, ter pogled, kjer lahko intuitivno in hitro oglas dodamo med priljubljene ali pa ga izbrišemo s seznama za nas relevantnih oglasov (v nadaljevanju kot "tinder"pogled). Pogleda si kombinirano izmenjujeta glavno vlogo - če pri prijavi v aplikacijo zaledni sistem vrne oglase, ki še niso bili pregledani, je primarni pogled tinder pogled in se tudi prikaže ob prijavi. Če ni zaznanih novih oglasov, se po prijavi pokaže pogled, v katerem je prikazan seznam vseh oglasov.

Tinder pogled

Tinder pogled (prikazan na sliki 3.14) je pogled, kjer simuliramo že obstoječo aplikacijo za spoznavanje in parčkanje imenovano Tinder [31], le da je prilagojena za naš uporabniški primer. Zgledovali smo se po njihovem izgledu, ki je sestavljen iz več kart, zloženih ena na drugo, vsako karto, ki predstavlja določen oglas, pa lahko enostavno s potegom prsta v levo ali desno dodamo med priljubljene oziroma odstranimo iz našega seznama oglasov (Slika 3.15). Poleg tega lahko s klikom na karto vidimo tudi detajle oglasa in se šele nato vrnemo nazaj na karto, ter jo ustrezno razvrstimo. Po razvrstitvi se na vrhu kupa prikaže naslednji oglas. Z drugimi funkcionalnostmi tega pogleda nismo obremenjevali, saj želimo to razvrščanje oziroma hitro dodajanje med priljubljene poenostaviti in narediti karseda intuitivno.

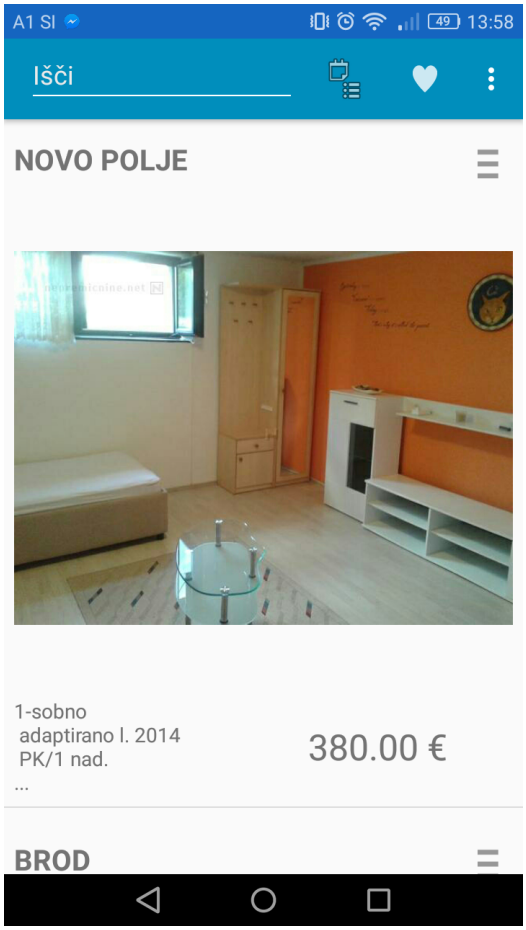


Slika 3.14: Tinder pogled mobilne aplikacije

Slika 3.15: Tinder pogled mobilne aplikacije, ko uporabnik oglas povleče v levo

Pogled s seznamom vseh oglasov

Pogled s seznamom vseh oglasov (prikazan na sliki 3.16) je malce bolj informativen. Omogoča namreč pregled vseh oglasov, ki so bili zaznani, in napredno iskanje ter filtriranje nad njimi. Že v seznamu se prikaže več detajlov kot v tinder pogledu, ob dotiku oglasa pa se odpre predogled celotnega oglasa. Omogoča več funkcionalnosti in manipulacijo oglasov, kot so na primer dodajanje med priljubljene, brisanje in razvrščanje. Vidimo lahko tudi komentarje, ki smo jih dodali prek spletne aplikacije ali pa prek razširitve v brskalniku (opisano v poglavju 3.5).



Slika 3.16: Pogled mobilne aplikacije s seznamom vseh oglasov

3.4.2 Razvoj mobilne aplikacije

Ker je eden izmed namenov razvijanja te celovite rešitve zmanjšati porabo časa, ki ga ljudje porabimo pri iskanju spletnih oglasov, pa tudi hitro obveščanje o novih oglasih, je bil razvoj mobilne aplikacije logična poteza. Izjemno veliko časa namreč preživimo na mobilnih telefonih, mobilne aplikacije pa so v zadnjih nekaj letih vedno bolj popularne - nekatere ocene kažejo, da se bo med leti 2016 in 2020 za približno trikrat povečal prihodek, ki se ustvari z mobilnimi aplikacijami [38].

Uporaba zunanjih knjižnic Volley in Glide

Mobilna aplikacija je eden izmed odjemalskih modulov celovite rešitve in se za pridobivanje vseh potrebnih podatkov, tako kot spletna aplikacija in ostali odjemalski moduli, povezuje prek vhodne točke v zaledni sistem. Za te klice prek REST protokola je bila uporabljena knjižnica Volley [33]. Poleg poenostavljenih klicev v omrežje omogoča tudi načrtovanje zahtevkov in več sočasnih klicev, ni pa primerna za večje prenose z omrežja, saj celotno stanje prenosa hrani v svojem spominu. Knjižnico smo bolj kot ne uporabili takšno kot je, morali pa smo sprogramirati lastne javanske objekte, ki predstavljajo zahtevke. Volley v tem smislu ponuja npr. JsonObjectRequest, StringObjectRequest in druge glede na to, kaj pričakujemo kot odgovor oziroma kaj pošiljamo. Ker knjižnica ne omogoča enostavnega dodajanja zaglavij (ang. headerjev), mi pa moramo zaradi zahtev vhodne točke v zaledni sistem k vsakemu zahtevku dodati avtentikacijski žeton, smo morali to sprogramirati sami. Tako smo uporabili javansko funkcionalnost razširjanja objektov in smo ustvarili dva nova objekta, ki sta razširila prej omenjena JsonObject- in StringObjectRequest (Koda 3.7). V teh smo prepisali metodo getHeaders, ki je tako avtomatsko pri vsaki uporabi teh objektov zahtevku dodala potreben žeton.

Koda 3.7: Razširjanje knjižnice Volley s svojimi objekti za zahtevke

```
1 public class DhJsonObjectRequest extends JsonObjectRequest {
2
3     private String bearerToken;
```

```
4       private Map<String, String> additionalHeaders;
5
6       public DhJsonObjectRequest(...) {
7           super(method, url, jsonRequest, listener, errorListener);
8           this.bearerToken = bearerToken;
9           if (additionalHeaders == null) {
10              this.additionalHeaders = new HashMap<>();
11          } else {
12              this.additionalHeaders = additionalHeaders;
13          }
14      }
15
16      @Override
17      public Map<String, String> getHeaders() throws AuthFailureError {
18          additionalHeaders.put("Authorization", "Bearer " + bearerToken);
19          return additionalHeaders;
20      }
21  }
```

Ker velik del mobilne aplikacije predstavljajo tudi slike oglasov, ki se morajo prenašati asinhrono, smo iskali dodatno knjižnico tudi za to funkcionalnost. Prej omenjena knjižnica Volley sicer omogoča asinhron prenos slik, vendar smo pri primerjanju Volley in Glide [12] knjižnic (pa tudi privzete Android AsyncTask funkcionalnosti) opazili, da je Glide veliko bolj enostavna, če želimo slike asinhrono dodajati v že izpisan seznam (ListView objekt v Android programiranju). Primerjava je prikazana v odseku kode, prikazane spodaj (Koda 3.8 in Koda 3.9).

Koda 3.8: Uporaba Glide knjižnice pri asinhronem dodajanju slike v seznam

```
1  Glide.with(tinderAdvertsView.getApplicationContext()).load(image).into(advertImageView);
```

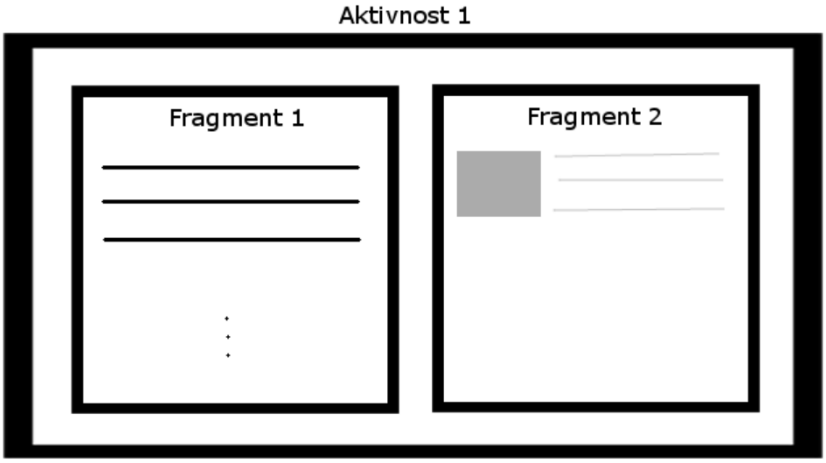
Koda 3.9: Uporaba privzete Android AsnycTask knjižnice pri asinhronem dodajanju slike v seznam

```
1
2  ImageLoaderAsync imageLoaderTask = new ImageLoaderAsync(getListView(), getActivity()
3      .getCacheDir());
4
5  HashMap<String, String> hmDownload = new HashMap<String, String>();
6  hmDownload.put("flag_path", imgUrl);
7  hmDownload.put("position", String.valueOf(i));
8
9  // Starting ImageLoaderTask to download and populate image in the listview
10 imageLoaderTask.execute(hmDownload);
```

Android fragmenti in aktivnosti

Vsak izmed prej omenjenih pogledov aplikacije ima še podpoglede (npr. seznam oglasov je en pogled, čemur sledi podroben prikaz oglasa kot podpogled). Najprej smo vsakega izmed teh pogledov in podpogledov implementirali v svojo aktivnost (ang. activity) in kasneje spoznali, da je to napačen pristop. Aktivnosti so pri izvajanju android aplikacije namreč drage (vsaka aktivnost ima več faz, shranjujejo se na sklad), prehod med njimi pa je zahteven, zato naj bi, kadarkoli je mogoče, uporabljali fragmente, aktivnosti pa le, ko res ni druge izbire [9].

Tako smo torej refaktorirali kodo in začeli z implementacijo fragmentov. Odločili smo se za pristop, kjer je en primer uporabe aplikacije ena aktivnost, sestavljena iz več fragmentov. Tako je na primer pregled oglasov v seznamu en fragment, predogled oglasa z osnovnimi podatki drug fragment, skupaj pa je to ena aktivnost (Slika 3.17).



Slika 3.17: Fragmenti aplikacije sestavljeni v eno aktivnost

3.5 Razširitev v spletnem brskalniku

Razširitev v spletnem brskalniku je še tretji izmed odjemalcev zaledne storitve naše celovite rešitve. Namenjen je predvsem enostavnemu dodajanju

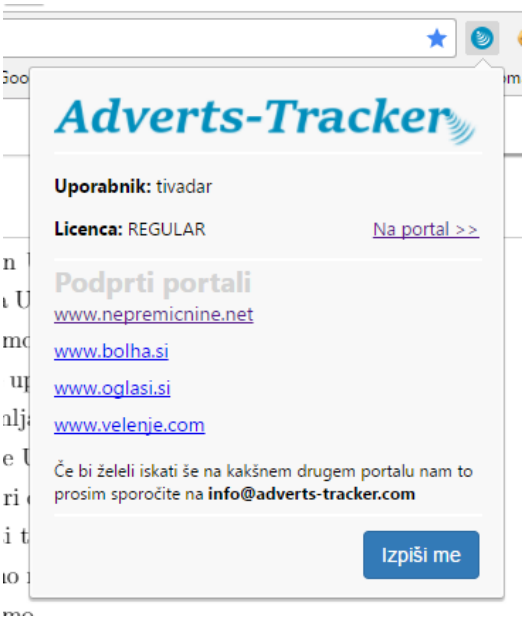
novih iskalnih kriterijev v algoritem obveščanja, kot tudi pomoči pri spremljanju oglasov na zunanjih portalih. Algoritem za obveščanje deluje na principu neposrednih URL naslovov do iskalnih kriterijev, kar je tudi glavni razlog za implementacijo razširitve. Če želimo uspešno slediti, moramo v podatkovni bazi imeti pravilno strukturiran URL naslov do rezultatov iskanja, kar pomeni, da bi ročno vnašanje tega URL naslova v aplikacijo za nevešče uporabnike bil (pre)velik izziv. Zato smo se odločili razviti razširitev, ki se namesti v brskalnik in ko ta zazna, da je uporabnik na spletni strani, ki jo naša rešitev podpira, ponudi možnost spremljanja oglasov. Dodajanje novih kriterijev je torej intuitivno, ročno kopiranje URL naslovov pa ni potrebno. Tako smo se izognili morebitnim napakam pri dodajanju novih kriterijev, cena tega pa je, da si mora uporabnik namestiti to razširitev. Alternativa tej rešitvi bi bila, da uporabniku vseeno ponudimo možnost ročnega vnosa URL naslova, ki ga pred shranjevanjem validiramo.

V času pisanja te diplomske naloge je bila implementirana razširitev za Google Chrome spletni brskalnik. Nadaljnje pisanje se navezuje na funkcionalnosti, ki jih lahko dosežemo z Google Chrome razširitvami.

3.5.1 Izgled

Po namestitvi se, kot za vse druge razširitve, v zgornjem desnem kotu brskalnika pokaže ikona naše aplikacije. Ker so vsi oglasi vezani na uporabnika, se mora uporabnik tudi v to razširitev (tako kot v vse druge odjemalske module) prijavit z uporabniškim imenom in geslom. Po prijavi je glavno okence razširitve zgolj informativne narave - prikazuje uporabniško ime, trenutno licenco in pa vse podprte portale. Na voljo je tudi hitra povezava do spletne aplikacije (Slika 3.18).

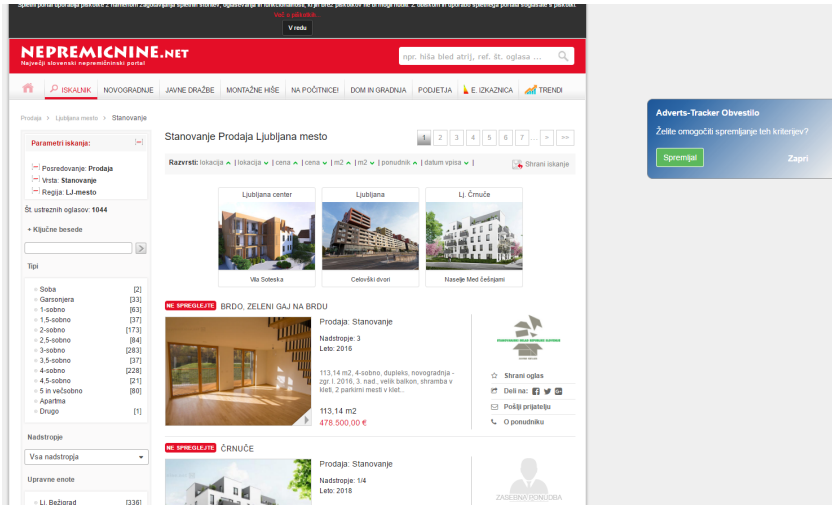
Glavna funkcionalnost se prikaže, ko uporabnik obišče eno izmed podprtih spletnih strani. Če razširitev zazna, da je na podprtem spletnem mestu in na prikazovanju oglasov, se ob strani prikaže relativno majhno, a vseeno vidno okence, ki uporabnika obvesti o možnosti sledenja oglasom. V tem okencu je tako možnost dodajanja sledenja (Slika 3.19), ki se potrdi v dveh



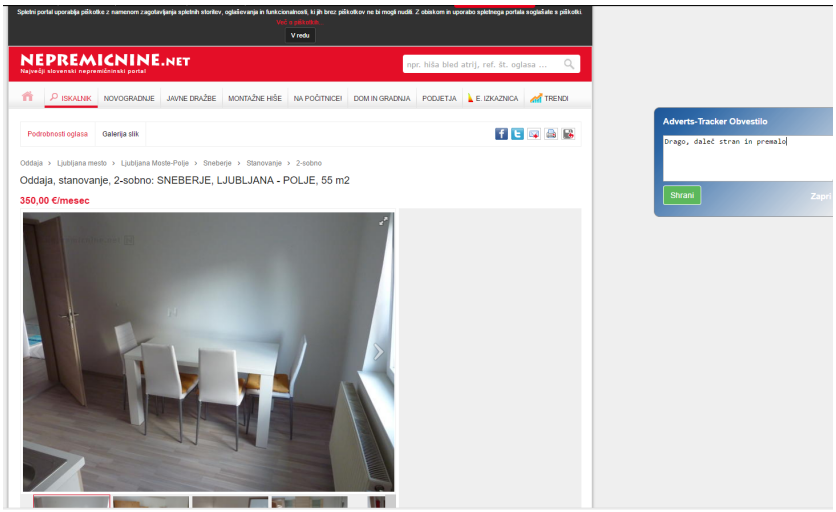
Slika 3.18: Informativno okence po prijavi v razširitev spletnega brskalnika

korakih. Najprej uporabnik klikne na gumb, da želi kriterijem slediti, nato pa v okence, ki se prikaže, vpiše še naziv za to iskanje. Ta naziv se uporablja v spletni aplikaciji za filtriranje ter v obvestilih, tako da uporabnik že iz obvestila ve, kje je bil nov oglas objavljen.

Stranska funkcionalnost razširitve je, da uporabniku omogoča enostavno dodajanje lastnih zapiskov k oglasu (Slika 3.20). Ko razširitev zazna, da je uporabnik na oglasu, ki je bil objavljen v njegov nabor kriterijev, prikaže okence, kamor lahko uporabnik doda lastne zapiske. Ti lastni zapiski se shranijo v podatkovno bazo in so na voljo tako v mobilni kot tudi spletni aplikaciji; prikažejo se tudi, če uporabnik še enkrat pride na ta oglas - v tem primeru je to vnosno polje za lastne zapiske predizpolnjeno s prejšnjimi zapiski.



Slika 3.19: Funkcionalnost razširitve spletnega brskalnika za dodajanja iskalnih kriterijev



Slika 3.20: Funkcionalnost razširitve spletnega brskalnika za dodajanja komentarjev na obstoječ oglas

3.5.2 Razvoj razširitve

Razvoj razširitev je s sintaktičnega vidika enak oziroma podoben razvoju kakršne koli spletne strani. Glavni programski jezik je JavaScript s poljubnimi zunanjimi knjižnicami, za izgled pa se uporablja HTML.

Struktura Google Chrome razširitev

Goole Chrome za implementacijo razširitev predvideva več nivojev izvajanja [16], vse te in še druge nastavitve pa se dodajajo v enotni datoteki imenovani manifest.json. Nivoji izvajanja so:

- izvajanje v ozadju (ang. background scripts),
- izvajanje v ospredju (ang. content scripts) in
- izvajanje v glavnem meniju (ang. popup scripts).

V ozadju se načeloma izvaja le ena skripta (ang. background script), v ospredju pa lahko za vsako spletno stran določimo svojo skripto izvajanja. V prej omenjeni datoteki manifest.json določimo, na katerih spletnih straneh se bo izvajala katera skripta v ospredju. Omogoča tudi nadomestne znake (ang. wildcards), konkretno z našega primera npr. „https://www.nepremicnine.net/oglasi-*“. Tako določimo, da se za vse strani, ki se začnejo s to predpono, izvaja enotna skripta (Koda 3.10).

Ker se vsaka skripta izvaja na svojem nivoju, komunikacija med njimi ni tako preprosta, da bi lahko klicali kar funkcije med sabo. Za komunikacijo Google Chrome omogoča uporabo chrome.runtime [14] (Koda 3.11) in chrome.tabs [15] (Koda 3.12) APIjev.

Koda 3.10: Definiranje različnih skript izvajanj na različnih nivojih v datoteki manifest.json

```
1
2  "content_scripts": [
3     {
4       "matches": [
5         "https://www.nepremicnine.net/oglasi-*"
6       ],
7       "js": [
```

```
8           "scripts/vendor/jquery-2.1.1.min.js",
9           "scripts/utils.js",
10          "scripts/general-on-site.js",
11          "scripts/nepremicnine/on-search.js"
12        ],
13        "css": [
14          "css/main.css"
15        ]
16      }
17  ]
18
19  "background": {
20      "scripts": [
21        "scripts/vendor/jquery-2.1.1.min.js",
22        "scripts/utils.js",
23        "scripts/background.js",
24        "scripts/app.js"
25      ],
26      "persistent": false
27  }
```

Koda 3.11: Komunikacija s skripto v ozadju

```
1  chrome.runtime.onMessage.addListener(function (request) {
2      var originator = request.originator;
3      var msg = request.msg;
4      console.log(request);
5      if (msg === "subscribe") {
6          DRIVERSHUB_BG.subscribe(originator, request.url, request.notes);
7      } else if (msg === "all_configurations") {
8          DRIVERSHUB_BG.getUserConfigurations();
9      } else if (msg === "go_to_dh") {
10         DRIVERSHUB_BG.goToDriversHub(request.startView);
11     } else if (msg === "read_advert_by_id") {
12         DRIVERSHUB_BG.readAdvertById(request.uniqueid);
13     } else if (msg === "save_user_notes") {
14         DRIVERSHUB_BG.setAdvertNotes(request.ad_id, request.user_notes);
15     }
16 });
```

Koda 3.12: Komunikacija s skripto v ospredju

```
1  chrome.tabs.query({active: true, currentWindow: true}, function (tabs) {
2      chrome.tabs.sendMessage(tabs[0].id, {
3          'originator': 'DH_BG',
4          'msg': 'all_configurations_answer',
5          'data': resp
6      }, function (response) {
7      });
8  });
```

Implementacija

Ker lahko za vsako spletno mesto z nadomestnimi znaki določimo svojo skripto v ospredju, smo se odločili, da implementacijo poenostavimo tako, da vse te skripte razdelimo po portalih. Tako imamo skripto, ki se izvaja na

portalu X, skripto, ki se izvaja na portalu Y, skripto, ki se izvaja na portalu Z in tako naprej. Zato nam pri programiranju ni potrebno preverjati, na kateri strani se nahajamo, saj lahko to enostavno predpostavljamo - če se nek blok kode izvede v neki skripti, vemo, da smo na želeni spletni strani.

Za izvajanje v ozadju imamo le eno skripto, ki je neodvisna od tega, na katerem spletnem mestu se nahajamo. Skrbi za komunikacijo z zalednim sistemom celovite rešitve. Vsi podatki, ki so vezani na svojo spletno stran, so vhodni podatki za to skripto. Konkretno to pomeni, da ne glede na to, na katerem portalu je uporabnik, če želi shraniti nov kriterij, skripta v ospredju pošlje vse parametre v skripto v ozadju, ta pa nato pokliče zaledni sistem celovite rešitve in shrani potrebne informacije.

Skripta v ospredju skrbi le za prikaz pravilnega prikaznega okna (okno za sledenje novemu kriteriju ali pa okno, ki se pokaže, če smo že na oglasu - torej okno za lastne zapiske). Vse podatke pridobi iz skripte v ozadju, s katero komunicira prek chrome.runtime.sendMessage APIja. API omogoča pošiljanje poljubno strukturiranih sporočil in tako omogoča, da jih v skripti v ozadju enostavno ločimo glede na to, s katere sprednje skripte je sporočilo prišlo (in tako torej na kateri portal je sporočilo vezano).

Na drugi strani skripta v ozadju naprej pošlje odgovor prek chrome.tabs.sendMessage APIja, ki podobno omogoča poljubno strukturirano sporočilo.

Konkreten primer izvajanja

Za lažjo predstavo delovanja in funkcionalnosti posameznih skript predstavljamo konkreten primer izvajanja.

Uporabnik gre na poljuben podprt portal in poklika želene kriterije ter pritisne gumb „išči“. Po kliku je preusmerjen na rezultat iskanja, ki je spletno mesto s specifičnim URL naslovom. Za ta URL naslov smo v manifest.json definirali svojo skripto izvajanja v ospredju. Izvajanje je sledeče:

1. Skripta v ospredju pošlje zahtevek za vse konfiguracije, saj želi izvedeti, ali temu kriteriju uporabnik že sledi. Ta zahtevek pošlje v skripto v

ozadju prek APIja google.runtime.sendMessage.

2. Skripta v ozadju iz lokalnega pomnilnika brskalnika (ang. chrome local storage) pridobi uporabnika in njegov avtentikacijski žeton. V lokalni pomnilnik so se ti podatki shranili ob prijavi v razširitev. Zahtevek, ki ga je prejela iz skripte v ospredju, skupaj z avtentikacijskim žetonom pošlje v zaledni sistem celovite rešitve. Ko pridobi odgovor, ga posreduje nazaj v ospredje prek chrome.tabs.sendMessage APIja.

3. Skripta v ospredju preveri vse prejete konfiguracije - če ugotovi, da to sledenje še ne obstaja, prikaže obvestilo z možnostjo dodajanja novega sledenja.

3.6 Postavitev aplikacije pred prve uporabnike ter prvi problemi

Celovito rešitev smo postavili pred prve uporabnike, saj smo želeli dobiti konstruktivne predloge in kritike, da bi implementacijo izboljšali na podlagi konkretnih argumentov. Z začetkom uporabe aplikacije so se pojavili tudi prve težave, do katerih pride pri postavitvi aplikacije v realni svet. Težave so v veliki meri nastale zaradi neizkušenosti, saj na nekatere stvari enostavno nismo bili pozorni pri programiranju ali izbiri ustrezne strojne opreme.

3.6.1 Odzivi prvih uporabnikov

Ideja in prva rešitev, ki je v tistem času nastala, je bila objavljena na več slovenskih forumih o avtomobilizmu in motorizmu, kot so www.motosvet.com, www.alfa-klub.com, www.bmwslo.com. Nekaj izmed odzivov je vidnih spodaj:

- „Izpolnil anketo. Res super aplikacija, konstantno uporabljam.“[41]

- „Na žalost ne uporabljam Chrom-a (imam raje FF) toda vseeno, odlična zadeva in predvsem zamisel. Sem ga preizkusil v Chromu (ga imama kakor back up) in deluje dobro

  Sedaj, če bi le še za FF naredil bi bilo super Bi ga pa prilagodil še za bolho ter tako omogočil slednje oglasom še tam.

  P.S. je dober tudi za spremljat oglase o plovilih.“[44]

- „Hvala, bom preveril zadevo. Bi šlo še kaka podpora za Bolha.com? Jaz sem do sedaj link s seznamom po željah dal kot bookmark v poseben folder, nato pa enkrat dnevno odprl cel folder bookmarkov naenkrat in šel gledat..“[43]

- „Vrhunsko!! Osebno bi si želel takšno aplikacijo tudi za moto.it kjer je ponudba neprimerno večja in sledljivost toliko težja. Kar tako naprej!“[45]

- „Res odlična ideja! Bravo!“[40]

- „Sprobal in deluje super! Res odlična ideja!“[39]

- „Zanimivo

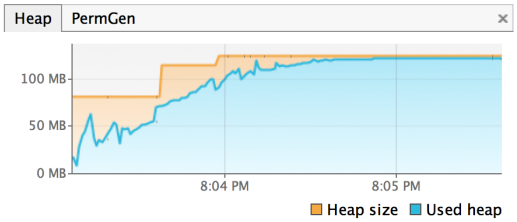
  Nekaj predlogov

  v opisu dodaj še letnik vozila... lahko tudi kilometre možnost sortiranja seznama po abecedi, ceni, času... ob ogledu oglasa imam nekaj sekund časa da ga shranim, nato izgine iz seznama. Ali je možnost oglas shraniti tudi ob kasnejšem ogledu?“[42]

3.6.2 Težave z velikostjo pomnilnika (ang. heap space)

Največja težava je bila, da je na strežniku po nekaj časa uporabe zmanjkalo medpomnilnika. Problem se je kazal tako, da strežnik, na katerem je bila gostovana spletna aplikacija, ni več uspel odgovarjati zahtevkom. V dnevniških datotekah (ang. log files) so bili zapisi tipa pomanjkanje spomina (ang. out of memory) in podobni. Zaradi tega smo se odločili, da bomo potrebo po

spominu začeli spremljati z orodjem VisualVM, opisanim v poglavju 2.5. Orodje smo namestili neposredno na produkcijski strežnik kot tudi na lokalno razvojno okolje. Opazili smo, da se poraba spomina občutno poveča že takoj po prijavi uporabnika v sistem, še dodatno pa pri sprehajanju po podstraneh aplikacije. Zaradi tega smo prišli do zaključka, da mora nekje obstajati puščanje pomnilnika (ang. memory leak), saj je tudi po obnavljanju pomnilniškega prostora (ang. garbage collection) zasedenost pomnilnika naraščala, nikoli pa se ni spustila na prvotno stanje (Slika 3.21).



Slika 3.21: Primer puščanja pomnilnika

Ker imamo pri celoviti rešitvi veliko klicev med odjemalskimi moduli in zalednim sistemom, smo se najprej osredotočili na ta nivo aplikacije in takoj opazili problem - nikjer namreč nismo poskrbeli, da se zapirajo povezave. Tako se je pri vsakem klicu naredila nova povezava, ki jo za sabo nismo počistili oziroma zaprli. Težavo smo odpravili, še vedno pa se je dogajal prvotni problem, le da sedaj v manjši obliki.

Ker torej problem še ni bil odpravljen, lokalno pa ga nismo uspeli reproducirati (vsaj ne v takšni meri), smo se odločili prekopirati tudi bazo podatkov iz okolja, ki so ga uporabljali prvi uporabniki. Po tem se je problem v isti meri kazal tudi na lokalnem okolju. Ugotovili smo, da so glavni problem slike - slike so se v podatkovni bazi shranjevale zakodirane po Base64 načinu in so se tako prikazovale tudi v spletni aplikaciji. Vse slike skupaj z ostalimi podatki so se torej neposredno shranjevale v medpomnilnik na spletnem strežniku, kar je posledično kmalu privedlo do opisanega problema. Rešitev je bila, da so se slike shranile v pomnilnik strežnika, na aplikaciji pa so se prikazovale kot pot do direktorija, kjer so se nahajale.

3.6.3 Težave s ponudnikom storitev v oblaku

Pri francoskem ponudniku storitev v oblaku smo najeli strežnik, ki se prodaja kot strojna oprema (ang. bare metal), kar pomeni, da ima vsak svojo dedicirano strojno opremo in ne virtualnega strežnika, kot je to v navadi pri večjih ponudnikih. Glavni problem pri tem ponudniku je bila predvsem nezanesljivost opreme in neodzivnost njihove podporne ekipe. Večkrat se je zgodilo, da se je strežnik enostavno ugasnil in odklopil od interneta, zaradi česar smo izgubili dostop in posledično celotno podatkovno bazo.

Postavitev novega strežnika je izjemno hitra, a brez obstoječe baze podatkov za končne uporabnike brezpredmetna. Rešitev problema bi bila, da bi se prestavili na novega ponudnika, a smo zaenkrat rešili le simptom - naredili smo skripto, ki vsak dan naredi kopijo podatkovne baze na drug strežnik in tako za polovico zmanjšali verjetnost izgube podatkov (Koda 3.13). Za popolno izgubo podatkov bi sedaj morala odpovedati oba strežnika naenkrat.

Koda 3.13: Skripta za periodično kopiranje podatkovne baze na drug strežnik

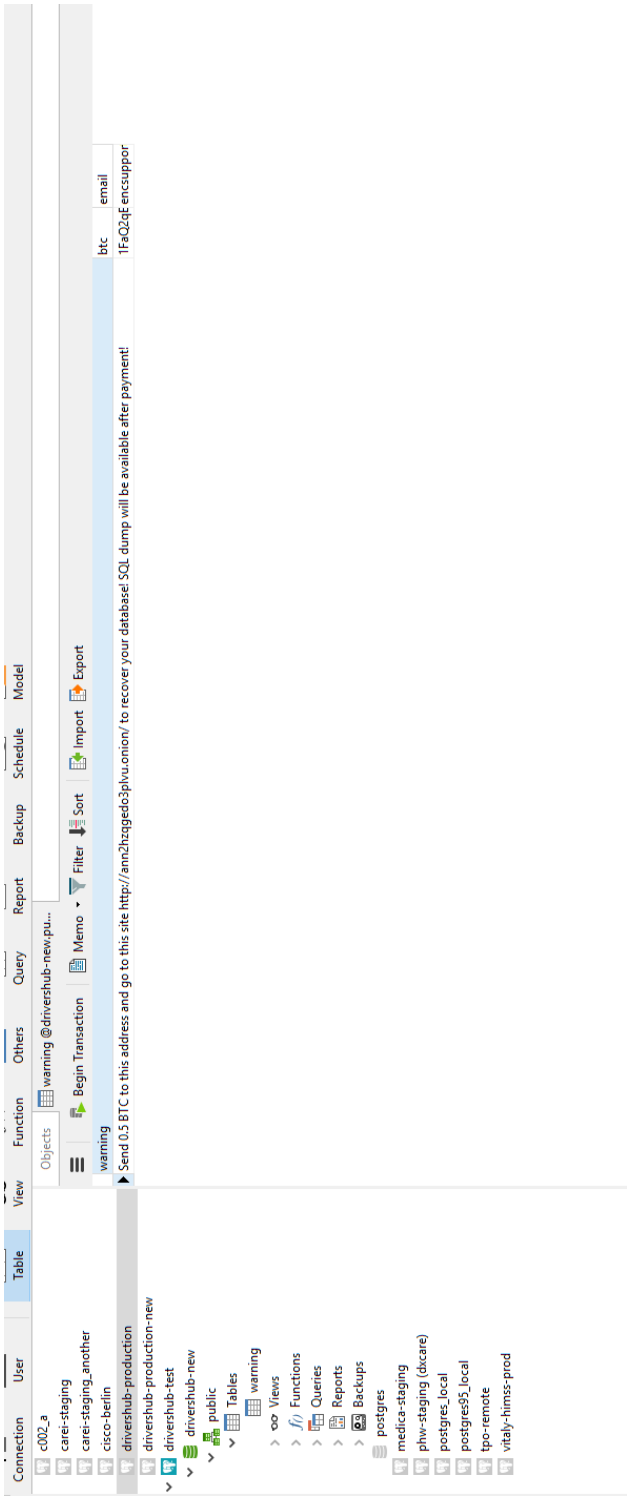
```
1  #!/bin/bash
2
3  action=$1
4
5  BACKUP_DIR=/opt/drivers-hub/DH_BCKP
6  DB_USER=username
7  DB_PASSWORD=password
8  REMOTE_IP=ip_address
9  REMOTE_PORT=port
10
11 function makebackup {
12  local date=$(date +"%Y%m%d%H%M")
13  pg_dump -U postgres -h 163.172.178.251 -p 5432 drivershub-new >> $BACKUP_DIR/backup_$date
14  echo Made db backup "${BACKUP_DIR}/backup_${date}" ...
15 }
16
17 function clearfolder {
18  find $BACKUP_DIR -mtime +10 -type f -delete
19  echo Cleared all backups older than 10 days, if there were any...
20 }
21
22 if [[ $action = "makebackup" ]]; then
23    makebackup
24 elif [[ $action = "clear" ]]; then
25    clearfolder; fi
```

3.6.4 Napad na podatkovno bazo

Kmalu po postavitvi aplikacije v javno okolje se je zgodil napad na podatkovno bazo. Pri odprtju povezave smo opazili, da so bile vse tabele izbrisane, ustvarjena pa je bila nova tabela, kjer je bilo zapisano, da je bila vsebina baze kriptirana. Za dekriptiranje so napadalci zahtevali plačilo v Bitcoin valuti (Slika 3.22).

Rešitev je bila enostavna, saj se je to zgodilo po tem, ko je bila postavljena skripta za kopiranje podatkovne baze. Vseeno pa je bilo treba najti luknjo, prek katere je napadalec prišel v sistem. Izkazalo se je, da je bila luknja več kot očitna - povezava do podatkovne baze je bila namreč javno dostopna, uporabniško ime in geslo pa nastavljeno na privzeto vrednost („postgres“, „postgres“). Predvidevamo, da je napadalec enostavno skeniral odprta vrata sistema in ker je podatkovna baza delovala na privzetih vratih, vstopil v sistem, naredil prepis podatkov (ang. dump) in ustvaril svojo tabelo.

Podatkovna baza seveda ne bi smela biti javno dostopna in za to tudi ni potrebe, kar se tiče delovanja aplikacije. Kot je vidno na sliki 3.1, je znotraj nedostopnega okvirja. Odprt mora biti le dostop za varnostno mikrostoritev in za vstopno točko v zaledni sistem. Razlog, zakaj smo imeli javno dostopno bazo je, da smo lahko do nje dostopali z odjemalci na naših računalnikih (npr. PgAdmin, Navicat). To smo sedaj rešili tako, da smo vzpostavili virtualno omrežje, na katerega se moramo najprej povezati, če želimo dostopati do podatkovne baze. Podatkovna baza pa je odprta le za to omrežje in ni več javno dostopna.



Slika 3.22: Posledica napada na podatkovno bazo

Poglavje 4

Sklep in zaključek

Skozi proces pisanja diplomske naloge smo si ogledali potek implementacije naše celovite rešitve, sestavljene iz spletne aplikacije, mobilne aplikacije, razširitve v spletnem brskalniku in zalednega sistema, ki vse to omogoča. Na začetku diplomske naloge smo si ogledali tehnologije, ki smo jih uporabili ter pristop, s katerim smo se soočili s posameznimi problemi. V nadaljnjih poglavjih smo se sprehodili vse od začetka razvoja ideje, načrtovanja arhitekturnega dela rešitve in podatkovnega modela ter do konca do same implementacije oziroma kodiranja posameznih odjemalskih modulov. Pri opisovanju implementacij posameznih delov smo pisali tako abstraktno - o miselnem procesu, na katerem temelji kakšen del izvajalne logike - pa tudi bolj konkretno, kot je argumentacija odsekov kode in praktična uporaba zunanjih knjižnic. Na koncu diplomskega dela smo pisali še o problemih, s katerimi smo se srečali, ko smo posamezni del celovite rešitve prvič postavili v javno dostopno okolje in pred prve uporabnike. Predvsem te težave so bile pokazatelj, kje se trenutno nahajamo v smislu pripravljenosti rešitve na večji nabor uporabnikov, še posebno kar se tiče skaliranja rešitve in pa približno kakšno obremenjenost arhitekturnega dela lahko pričakujemo, ko število uporabnikov povečamo. Poleg takšnega neposrednega odziva samega okolja pa smo dobili tudi enako pomembne odzive prvih uporabnikov, na podlagi katerih smo ustrezno spreminjali predvsem uporabniški vmesnik ter

dodajali in tudi umikale nekatere funkcionalnosti.

Splošen odziv prvih uporabnikov je bil izjemno pozitiven, a je to vseeno prej izjema kot pravilo pri takšnih aplikacijskih rešitvah. K problemu bi bilo že takoj na začetku potrebno pristopiti na drugačen način. Stvari, kot sta želja po različnih funkcionalnostih in potreba po takšni aplikaciji (in druge podobne) so stvari, ki bi se skozi realen proces implementacije takšne rešitve preverjale pred dejanskim začetkom kodiranja, saj so pivoti, ki se izvajajo tako pozno v procesu finančno preveč obremenjujoči. Tako bi tudi mi, v primeru ponovne izvedbe takšne rešitve (in pod drugačnimi pogoji), predhodno naredili temeljno analizo trga, s katero bi preverili potrebo po specifičnih funkcionalnostih in že k načrtovanju pristopili drugače. Poleg tega bi tako pridobili temeljni načrt razvoja oziroma specifikacijo posameznih uporabniških zgodb, na podlagi katerih bi najprej naredili okvirno sliko celotne programske logike. Tako bi se izognili potrebi po večjem refaktoriranju kode in spreminjaju že sprogramiranih stvari tako pozno v procesu.

Z vidika uporabljenih tehnologij menimo, da smo izbrali primeren nabor, ki smo ga v zato namenjenem poglavju diplomske naloge tudi argumentirali. To seveda ne pomeni, da bi bil drugačen izbor slabši ali boljši, vseeno pa bi zaradi okornosti JSF in preizkušanja alternativ ter skladnosti s trenutnimi trendi mogoče implementacijo spletne aplikacije z JSF tehnologije prestavili na kakšno izmed odjemalskih tehnologij, kot so AngularJS [2], ReactJS [27], Vue [34] in podobne.

Cilj diplomske naloge, ki je bil razviti celovito rešitev, sestavljeno iz več odjemalskih modulov, hkrati pa se osredotočati na zaledni sistem, je izpolnjen. Produkt, ki je nastal predstavlja, kako z javanskimi mikrostoritvami izdelati kompleksen zaledni sistem, sestavljen iz več modulov, ki so med seboj neodvisni in omogočajo samostojno skaliranje. Prav tako dobro prikazuje tudi drugo pozitivno plat mikrostoritev, ki je neljuba, a pri takšnem manjšem projektu zelo verjetna posledica - in to je potreba po refaktoriranju posameznih delov kode, brez da bi s tem posegali v samo jedro potencialnega monolita.

Razvili smo celovito rešitev in se skozi celoten proces od začetka do konca ogromno naučili. Naučili smo se novih tehnologij, načrtovanja podatkovnega modela, postavitve celotne rešitve na aplikacijski strežnik, osnovne konfiguracije Linux operacijskega sistema in omrežnih nastavitev, kako sodelovati s prvimi uporabniki in tudi kako pristopati do ponudnika storitev v oblaku, ko gre kaj narobe. Največ pa smo se naučili skozi proces samega pisanja diplomske naloge, ko smo implementacijo rešitve pretvorili v besede na papirju in se skozi retrospektivo še enkrat soočili z vsemi problemi - tokrat v drugačnem kontekstu in s širšo, bolj jasno sliko v mislih.

Literatura

[1] Android studio: Preview. Dosegljivo: https://developer.android.com/studio/preview/index.html, 2017. [Dostopano: 28. 06. 2017].

[2] Angularjs. Dosegljivo: https://angularjs.org/, 2017. [Dostopano: 28. 06. 2017].

[3] Apache camel. Dosegljivo: http://camel.apache.org/, 2017. [Dostopano: 28. 06. 2017].

[4] Autobingooo. Dosegljivo: https://www.bingooo.com, 2017. [Dostopano: 28. 06. 2017].

[5] Autoscout. Dosegljivo: https://www.autoscout24.it, 2017. [Dostopano: 28. 06. 2017].

[6] Bootcards - cards-based ui with dual-pane capability. Dosegljivo: http://bootcards.org/, 2017. [Dostopano: 28. 06. 2017].

[7] Bootstrap - js framework for developing responsive projects on the web. Dosegljivo: http://getbootstrap.com/, 2017. [Dostopano: 28. 06. 2017].

[8] Dropwizard: About. Dosegljivo: http://www.dropwizard.io/1.1.0/docs/about/index.html, 2017. [Dostopano: 28. 06. 2017].

[9] Fragments. Dosegljivo: https://developer.android.com/guide/components/fragments.html, 2017. [Dostopano: 28. 06. 2017].

[10] Git. Dosegljivo: https://git-scm.com/, 2017. [Dostopano: 28. 06. 2017].

[11] Github. Dosegljivo: https://github.com/, 2017. [Dostopano: 28. 06. 2017].

[12] Glide: Displaying images with the glide library. Dosegljivo: https://github.com/codepath/android_guides/wiki/Displaying-Images-with-the-Glide-Library, 2017. [Dostopano: 28. 06. 2017].

[13] Google analytics. Dosegljivo: https://www.google.com/analytics, 2017. [Dostopano: 28. 06. 2017].

[14] Google chrome chrome.runtime api. Dosegljivo: https://developer.chrome.com/extensions/runtime, 2017. [Dostopano: 28. 06. 2017].

[15] Google chrome chrome.tabs api. Dosegljivo: https://developer.chrome.com/extensions/tabs, 2017. [Dostopano: 28. 06. 2017].

[16] Google chrome extensions developer's guide. Dosegljivo: https://developer.chrome.com/extensions/devguide, 2017. [Dostopano: 28. 06. 2017].

[17] The history of java technology. Dosegljivo: http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html, 2017. [Dostopano: 28. 06. 2017].

[18] Jetty. Dosegljivo: http://www.eclipse.org/jetty/, 2017. [Dostopano: 28. 06. 2017].

[19] jsoup: Java html parser. Dosegljivo: https://jsoup.org/, 2017. [Dostopano: 28. 06. 2017].

[20] jsoup: Java server faces technology. Dosegljivo: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html, 2017. [Dostopano: 28. 06. 2017].

[21] Keycloak: About. Dosegljivo: http://www.keycloak.org/about.html, 2017. [Dostopano: 28. 06. 2017].

[22] Kumuluzee. Dosegljivo: https://ee.kumuluz.com/, 2017. [Dostopano: 28. 06. 2017].

[23] Maven: What is maven. Dosegljivo: https://maven.apache.org/what-is-maven.html, 2017. [Dostopano: 28. 06. 2017].

[24] Motosvet forum, objava uporabnika o morebitnih informacijah za željen oglas. Dosegljivo: http://motosvet.com/tabla/topic/34559-oglasi-iz-interneta/?do=findComment&comment=2356337, 2017. [Dostopano: 29. 06. 2017].

[25] Postgresql. Dosegljivo: https://www.postgresql.org/, 2017. [Dostopano: 28. 06. 2017].

[26] Quartz job scheduler. Dosegljivo: http://www.quartz-scheduler.org/, 2017. [Dostopano: 28. 06. 2017].

[27] React - a javascript library for building user interfaces. Dosegljivo: https://facebook.github.io/react/, 2017. [Dostopano: 28. 06. 2017].

[28] Spark java. Dosegljivo: http://sparkjava.com/, 2017. [Dostopano: 28. 06. 2017].

[29] sql2o. Dosegljivo: http://www.sql2o.org/, 2017. [Dostopano: 28. 06. 2017].

[30] Sun acquisition by oracle. Dosegljivo: https://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle, 2017. [Dostopano: 28. 06. 2017].

[31] Tinder. Dosegljivo: https://www.gotinder.com/, 2017. [Dostopano: 28. 06. 2017].

[32] Visualvm. Dosegljivo: https://visualvm.github.io/, 2017. [Dostopano: 28. 06. 2017].

[33] Volley: Transmitting network data using volley. Dosegljivo: https://developer.android.com/training/volley/index.html, 2017. [Dostopano: 28. 06. 2017].

[34] Vue - the progressive javascript framework. Dosegljivo: https://vuejs.org/, 2017. [Dostopano: 28. 06. 2017].

[35] Why cards are the future of the web. Dosegljivo: https://blog.intercom.com/why-cards-are-the-future-of-the-web/, 2017. [Dostopano: 28. 06. 2017].

[36] Wildfly: About. Dosegljivo: http://wildfly.org/about/, 2017. [Dostopano: 28. 06. 2017].

[37] Wildfly swarm. Dosegljivo: http://wildfly-swarm.io/, 2017. [Dostopano: 28. 06. 2017].

[38] Worldwide mobile app revenues in 2015, 2016 and 2020. Dosegljivo: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/, 2017. [Dostopano: 28. 06. 2017].

[39] Blindek. Motosvet forum, odziv uporabnika na prvotno rešitev. Dosegljivo: http://motosvet.com/tabla/topic/39998-avtonet-tracker/?do=findComment&comment=2227921, 2016. [Dostopano: 03. 07. 2017].

[40] Dog66. Motosvet forum, odziv uporabnika na prvotno rešitev. Dosegljivo: http://motosvet.com/tabla/topic/39998-avtonet-tracker/?do=findComment&comment=2227914, 2016. [Dostopano: 03. 07. 2017].

[41] Hranata. Alfa forum, odziv uporabnika na prvotno rešitev. Dosegljivo: http://forum.alfa-klub.com/viewtopic.php?f=7&t=38030&

sid=629b631580ac428b787a7ebd535d4fe5#p502274, 2016. [Dostopano: 03. 07. 2017].

[42] Lunatic. Motosvet forum, odziv uporabnika na prvotno rešitev. Dosegljivo: http://motosvet.com/tabla/topic/39998-avtonet-tracker/?do=findComment&comment=2226475, 2016. [Dostopano: 03. 07. 2017].

[43] Nixxon. Bmw forum, odziv uporabnika na prvotno rešitev. Dosegljivo: https://www.bmwslo.com/topic/127326-ads-tracker-sledilec-oglasom-na-internetu/#entry2632131, 2016. [Dostopano: 03. 07. 2017].

[44] Sorgo. Bmw forum, odziv uporabnika na prvotno rešitev. Dosegljivo: https://www.bmwslo.com/topic/127326-ads-tracker-sledilec-oglasom-na-internetu/#entry2631677, 2016. [Dostopano: 03. 07. 2017].

[45] Sorty. Motosvet forum, odziv uporabnika na prvotno rešitev. Dosegljivo: http://motosvet.com/tabla/topic/39998-avtonet-tracker/?do=findComment&comment=2226305, 2016. [Dostopano: 03. 07. 2017].

Dodatki

Dodatek A

Anketa potencialnih uporabnikov

ANALIZA - Sumarnik

XSPOL	Spol:				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Moški)	157	90%	90%	90%
	2 (Ženski)	18	10%	10%	100%
Veljavni	Skupaj	175	100%	100%	
		Povprečje	1.1	Std. Odklon	0.3

XSTAR2a4	V katero starostno skupino spadate?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (do 20 let)	8	5%	5%	5%
	2 (21 - 40 let)	132	75%	75%	80%
	3 (41 - 60 let)	35	20%	20%	100%
	4 (61 let ali več)	0	0%	0%	100%
Veljavni	Skupaj	175	100%	100%	
		Povprečje	2.2	Std. Odklon	0.5

Q1	Ste v zadnjih 5 letih kupili rabljeno vozilo?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	145	83%	83%	83%
	2 (Ne)	30	17%	17%	100%
Veljavni	Skupaj	175	100%	100%	

Q2	Koliko rabljenih vozil ste v zadnjih petih letih kupili?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (1-2)	130	74%	79%	79%
	2 (3-5)	29	17%	18%	96%
	3 (6-10)	4	2%	2%	99%
	4 (11 ali več)	2	1%	1%	100%
Veljavni	Skupaj	165	94%	100%	
		Povprečje	1.3	Std. Odklon	0.6

Q3	Ste pri iskanju rabljenega vozila uporabili spletne oglase (npr. avto.net, bolha.com)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	167	95%	96%	96%
	2 (Ne)	7	4%	4%	100%
Veljavni	Skupaj	174	99%	100%	
		Povprečje	1.0	Std. Odklon	0.2

Q4	Ste pri iskanju rabljenega vozila uporabili tuje ponudnike spletnih oglasov (npr. mobile.de, subito.it)?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa
	1 (Da)	97	55%	56%	56%
	2 (Ne)	75	43%	44%	100%
Veljavni	Skupaj	172	98%	100%	
		Povprečje	1.4	Std. Odklon	0.5

Q6	Ali ste oglase za rabljena vozila pregledovali tudi preko pametnega telefona ali tablice?				
	Odgovori	Frekvenca	Odstotek	Veljavni	Kumulativa

	1 (Da)	126	72%	72%	72%
	2 (Ne)	48	27%	28%	100%
Veljavni	Skupaj	174	99%	100%	
		Povprečje	1.3	Std. Odklon	0.4

Q7	V času, ko ste kupovali rabljeno vozilo, kolikokrat na dan ste v povprečju pregledovali spletne oglase?				
	Odgovori	**Frekvenca**	**Odstotek**	**Veljavni**	**Kumulativa**
	1 (Enkrat na dan)	51	29%	29%	29%
	2 (Dvakrat na dan)	53	30%	31%	60%
	4 (Tri do petkrat na dan)	36	21%	21%	81%
	3 (Več kot petkrat na dan)	33	19%	19%	100%
Veljavni	Skupaj	173	99%	100%	
		Povprečje	2.3	Std. Odklon	1.1

Q8	Koliko časa ste v povprečju porabili za posamezno pregledovanje oglasov?				
	Odgovori	**Frekvenca**	**Odstotek**	**Veljavni**	**Kumulativa**
	1 (1-2 minuti)	23	13%	13%	13%
	2 (3-5 minut)	54	31%	31%	45%
	3 (6-10 minut)	41	23%	24%	68%
	4 (11-15 minut)	17	10%	10%	78%
	5 (Več kot 15 minut)	38	22%	22%	100%
Veljavni	Skupaj	173	99%	100%	
		Povprečje	3.0	Std. Odklon	1.4

Q9	Ste kdaj kupili aplikacijo za pametni telefon ali tablico?				
	Odgovori	**Frekvenca**	**Odstotek**	**Veljavni**	**Kumulativa**
	1 (Da)	10	6%	21%	21%
	2 (Ne)	37	21%	79%	100%
Veljavni	Skupaj	47	27%	100%	
		Povprečje	1.8	Std. Odklon	0.4